JBoss Enterprise Application Platform 5
JBoss Messaging Benutzerhandbuch

---

zur Verwendung mit der JBoss Enterprise Application Platform 5
Ausgabe 5.1.0

Tim Fox
Clebert Suconic
Jeff Mesnil
Howard Gao
Andy Taylor

JBoss Enterprise Application Platform 5 JBoss Messaging Benutzerhandbuch

---

zur Verwendung mit der JBoss Enterprise Application Platform 5 Ausgabe 5.1.0

Tim Fox

Jeff Mesnil

Andy Taylor

Clebert Suconic

Howard Gao

Herausgegeben von

Laura Bailey

Jared Morgan

Rechtlicher Hinweis

Copyright © 2011 Red Hat, Inc.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Zusammenfassung

Ein Handbuch zur Benutzung von JBoss Messaging 1.4 mit der JBoss Enterprise Application Platform 5.0 und deren Patch Releases.

Inhaltsverzeichnis

Vorwort **4**
1. Dokumentkonventionen 4
1.1. Typografische Konventionen 4
1.2. Konventionen für Seitenansprachen 5
1.3. Anmerkungen und Warnungen 6
2. Hilfe bekommen und Feedback geben 6
2.1. Brauchen Sie Hilfe? 6
2.2. Wir freuen uns auf Ihr Feedback! 7

Kapitel 1. Über JBoss Messaging 1.4 . . . . . **8**

Kapitel 2. Einführung **9**
2.1. JBoss Messaging Features 9
2.2. Kompatibilität mit JBossMQ 10
2.3. Von JBoss Messaging verwendete System-Properties 10
2.3.1. support.bytesId 11
2.3.2. retain.oldxabehaviour 11

Kapitel 3. JBoss Messaging Installation **12**

Kapitel 4. Ausführen der Beispiele **13**

Kapitel 5. Konfiguration **15**
5.1. Konfiguration von ServerPeer 15
5.2. ServerPeer-Attribute 17
5.2.1. ServerPeer Methoden 20
5.3. Die Datenbank wechseln 22
5.4. Konfiguration des Post Office 23
5.4.1. PostOffice-Attribute 26
5.5. Konfiguration des Persistenz-Managers 28
5.5.1. PersistenceManager MBean-Attribute 30
5.6. Konfiguration des JMS Nutzer-Managers 31
5.6.1. JMSUserManager gemanagte Bean-Attribute 32
5.7. Konfiguration von Destinationen 32
5.7.1. Vorkonfigurierte Destinationen 32
5.7.2. Konfiguration von Warteschlangen 33
5.7.2.1. Warteschlangen MBean-Attribute 33
5.7.2.1.1. Destination-Sicherheitskonfiguration 35
5.7.2.1.2. Destination Paging Parameter 35
5.7.2.1.3. Warteschlangen-gemanagte Bean Operationen 36
5.7.3. Konfiguration von Topics 37
5.7.3.1. Topic gemanagte Bean-Attribute 37
5.7.3.1.1. Destination-Sicherheitskonfiguration 38
5.7.3.1.2. Destination Paging Parameter 39
5.7.3.2. Topic gemanagte Bean Operationen 40
5.8. Konfiguration von "Connection Factories" 40
5.8.1. ConnectionFactory gemanagte Bean Attribute 43
5.9. Konfiguration des Remoting Connectors 45
5.10. ServiceBindingManager 50

Kapitel 6. Clustering Hinweise **51**
6.1. Eindeutige Server Peer ID 51
6.2. Geclusterte Destinationen 51

6.3. Geclusterte dauerhafte Abonnements 51
6.4. Geclusterte temporäre Destinationen 51
6.5. Nicht geclusterte Server 51
6.6. Nachrichtenreihenfolge im Cluster 52
6.7. Idempotente Operationen 52
6.8. Geclusterte Connection Factories 52

Kapitel 7. JBoss Messaging XA Recovery-Konfiguration 53

Kapitel 8. JBoss Messaging Message Bridge Konfiguration 54
8.1. Message Bridge Übersicht 54
8.2. Bridge-Deployment 55
8.3. Bridge-Konfiguration 55

Kapitel 9. Aktivieren der JBoss Messaging Ordering Group 61
9.1. Wie "Message Ordering Group" aktiviert wird 61
9.1.1. Aktivierung mit dem API 61
9.1.2. Konfigurationsänderungen 62
9.2. Hinweise und Einschränkungen 63

Änderungsverzeichnis 64

Vorwort

1. Dokumentkonventionen

Dieses Handbuch verwendet mehrere Konventionen, um bestimmte Wörter und Sätze hervorzuheben und Aufmerksamkeit auf bestimmte Informationen zu lenken.

In PDF- und Papierausgaben verwendet dieses Handbuch Schriftbilder des Liberation-Fonts-Sets. Das Liberation-Fonts-Set wird auch für HTML-Ausgaben verwendet, falls es auf Ihrem System installiert ist. Falls nicht, werden alternative, aber äquivalente Schriftbilder angezeigt. Beachten Sie: Red Hat Enterprise Linux 5 und die nachfolgende Versionen beinhalten das Liberation-Fonts-Set standardmäßig.

1.1. Typografische Konventionen

Es werden vier typografische Konventionen verwendet, um die Aufmerksamkeit auf bestimmte Wörter und Sätze zu lenken. Diese Konventionen und die Umstände, unter denen sie auftreten, sind folgende:

`Nichtproportional Fett`

Dies wird verwendet, um Systemeingaben hervorzuheben, einschließlich Shell-Befehle, Dateinamen und -pfade. Es wird ebenfalls zum Hervorheben von Tasten und Tastenkombinationen verwendet. Zum Beispiel:

> Um den Inhalt der Datei **`my_next_bestselling_novel`** in Ihrem aktuellen Arbeitsverzeichnis zu sehen, geben Sie den Befehl **`cat my_next_bestselling_novel`** in den Shell-Prompt ein und drücken Sie **`Enter`**, um den Befehl auszuführen.

Das oben aufgeführte Beispiel beinhaltet einen Dateinamen, einen Shell-Befehl und eine Taste. Alle werden nichtproportional fett dargestellt und alle können, dank des Kontextes, leicht unterschieden werden.

Tastenkombinationen unterscheiden sich von einzelnen Tasten durch das Pluszeichen, das die einzelnen Teile einer Tastenkombination miteinander verbindet. Zum Beispiel:

> Drücken Sie **`Enter`**, um den Befehl auszuführen.
>
> Drücken Sie **`Strg`**+**`Alt`**+**`F2`**, um zu einem virtuellen Terminal zu wechseln.

Das erste Beispiel hebt die zu drückende Taste hervor. Das zweite Beispiel hebt eine Tastenkombination hervor: eine Gruppe von drei Tasten, die gleichzeitig gedrückt werden müssen.

Falls Quellcode diskutiert wird, werden Klassennamen, Methoden, Funktionen, Variablennamen und Rückgabewerte, die innerhalb eines Abschnitts erwähnt werden, wie oben gezeigt **`nichtproportional fett`** dargestellt. Zum Beispiel:

> Zu dateiverwandten Klassen zählen **`filesystem`** für Dateisysteme, **`file`** für Dateien und **`dir`** für Verzeichnisse. Jede Klasse hat ihren eigenen Satz an Berechtigungen.

Proportional Fett

Dies kennzeichnet Wörter oder Sätze, die auf einem System vorkommen, einschließlich Applikationsnamen, Text in Dialogfeldern, beschriftete Schaltflächen, Bezeichnungen für Auswahlkästchen und Radio-Buttons, Überschriften von Menüs und Untermenüs. Zum Beispiel:

> Wählen Sie **System** → **Einstellungen** → **Maus** in der Hauptmenüleiste aus, um die **Mauseinstellungen** zu öffnen. Wählen Sie im Reiter **`Tasten`** auf das Auswahlkästchen **`Mit links bediente Maus`** und anschließend auf **`Schließen`**, um die primäre Maustaste von der linken auf die rechte Seite zu ändern (d.h., um die Maus auf Linkshänder anzupassen).
>
> Um ein Sonderzeichen in eine **gedit**-Datei einzufügen, wählen Sie **Anwendungen** → **`Zubehör`** → **`Zeichentabelle`** aus der Hauptmenüleiste. Wählen Sie als Nächstes **Suchen**

→ **Suchen** aus der Menüleiste der **Zeichentabelle**, geben Sie im Feld **Suchbegriff** den Namen des Zeichens ein und klicken Sie auf **Weitersuchen**. Das gesuchte Zeichen wird daraufhin in der **Zeichentabelle** hervorgehoben. Doppelklicken Sie auf dieses hervorgehobene Zeichen, um es in das Feld **Zu kopierender Text** zu übernehmen und klicken Sie anschließend auf die Schaltfläche **Kopieren**. Gehen Sie nun zurück in Ihr Dokument und wählen Sie **Bearbeiten** → **Einfügen** aus der **gedit**-Menüleiste.

Der oben aufgeführte Text enthält Applikationsnamen, systemweite Menünamen und -elemente, applikationsspezifische Menünamen sowie Schaltflächen und Text innerhalb einer grafischen Oberfläche. Alle werden proportional fett dargestellt und sind anhand des Kontextes unterscheidbar.

Nichtproportional Fett Kursiv oder ***Proportional Fett Kursiv***

Sowohl bei nichtproportional fett als auch bei proportional fett weist ein zusätzlicher Kursivdruck auf einen ersetzbaren oder variablen Text hin. Kursivdruck kennzeichnet Text, der nicht wörtlich eingeben wird, oder angezeigten Text, der sich abhängig von den gegebenen Umständen unterscheiden kann. Zum Beispiel:

Um sich mit einer Remote-Maschine via SSH zu verbinden, geben Sie an einem Shell-Prompt **ssh *username@domain.name*** ein. Falls die Remote-Maschine **example.com** ist und Ihr Benutzername auf dieser Maschine John lautet, geben Sie also **ssh john@example.com** ein.

Der Befehl **mount -o remount *file-system*** hängt das angegebene Dateisystem wieder ein. Um beispielsweise das **/home**-Dateisystem wieder einzuhängen, verwenden Sie den Befehl **mount -o remount /home**.

Um die Version des derzeit installierten Pakets zu sehen, verwenden Sie den Befehl **rpm -q *package***. Die Ausgabe sieht wie folgt aus: ***package-version-release***.

Beachten Sie die kursiv dargestellten Begriffe oben — username, domain.name, file-system, package, version und release. Jedes Wort ist ein Platzhalter entweder für Text, den Sie für einen Befehl eingeben, oder für Text, der vom System angezeigt wird.

Neben der Standardbenutzung für die Darstellung des Titels eines Werks zeigt der Kursivdruck auch die erstmalige Verwendung eines neuen und wichtigen Begriffs an. Zum Beispiel:

Publican ist ein *DocBook* Publishing-System.

1.2. Konventionen für Seitenansprachen

Ausgaben des Terminals und Auszüge aus dem Quellcode werden visuell vom umliegenden Text hervorgehoben durch sogenannte Seitenansprachen (auch Pull-Quotes genannt).

Eine an das Terminal gesendete Ausgabe wird in den Schrifttyp **nichtproportional Roman** gesetzt und wie folgt dargestellt:

```
books        Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

Auszüge aus dem Quellcode werden ebenfalls in den Schrifttyp **nichtproportional Roman** gesetzt, doch wird zusätztlich noch die Syntax hervorgehoben:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
   public static void main(String args[])
       throws Exception
   {
      InitialContext iniCtx = new InitialContext();
      Object         ref    = iniCtx.lookup("EchoBean");
      EchoHome       home   = (EchoHome) ref;
      Echo           echo   = home.create();

      System.out.println("Created Echo");

      System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
   }
}
```

1.3. Anmerkungen und Warnungen

Zu guter Letzt verwenden wir drei visuelle Stile, um die Aufmerksamkeit auf Informationen zu lenken, die andernfalls vielleicht übersehen werden könnten.

Anmerkung

Eine Anmerkung ist ein Tipp, ein abgekürztes Verfahren oder ein alternativer Ansatz für die vorliegende Aufgabe. Das Ignorieren von Anmerkungen sollte keine negativen Auswirkungen haben, aber Sie verpassen so vielleicht einen Trick, der Ihnen das Leben vereinfachen könnte.

Wichtig

Die Wichtig-Schaukästen lenken die Aufmerksamkeit auf Dinge, die sonst leicht übersehen werden können: Konfigurationsänderungen, die nur für die aktuelle Sitzung gelten oder Dienste, für die ein Neustart nötig ist, bevor eine Aktualisierung wirksam wird. Das Ignorieren von Wichtig-Schaukästen würde keinen Datenverlust verursachen, kann aber unter Umständen zu Ärgernissen und Frustration führen.

Warnung

Eine Warnung sollte nicht ignoriert werden. Das Ignorieren von Warnungen führt mit hoher Wahrscheinlichkeit zu Datenverlust.

2. Hilfe bekommen und Feedback geben

2.1. Brauchen Sie Hilfe?

Falls Sie Schwierigkeiten mit einer der in diesem Handbuch beschriebenen Prozeduren haben, besuchen Sie das Red Hat Kundenportal unter http://access.redhat.com. Via Kundenportal können Sie:

- eine Knowledgebase bestehend aus Artikeln rund um technischen Support für Red Hat Produkte durchsuchen oder zu durchstöbern.
- einen Support-Case bei Red Hat Global Support Services (GSS) einreichen.
- auf weitere Produktdokumentationen zugreifen.

Red Hat unterhält außerdem eine Vielzahl von Mailing-Listen zur Diskussion über Red Hat Software und Technologie. Eine Übersicht der öffentlich verfügbaren Listen finden Sie unter https://www.redhat.com/mailman/listinfo. Klicken Sie auf den Namen einer Liste für weitere Einzelheiten zum Abonnieren dieser Liste oder um auf deren Archiv zuzugreifen.

2.2. Wir freuen uns auf Ihr Feedback!

Falls Sie einen Tippfehler finden oder eine Idee zur Verbesserung des Handbuchs \nhaben, freuen wir uns über Ihr Feedback! \nBitte reichen Sie hierzu einen Fehlerbericht für das Produkt **JBoss Enterprise Application Platform 5** und die Komponente **doc-JBoss_Messaging_User_Guide** ein. \nDas folgende Link bringt Sie zu einem bereits ausgefüllten Bug-Report für dieses Produkt: http://bugzilla.redhat.com/.

Füllen Sie folgende Vorlage in Bugzillas **Description**-Feld aus. Bitte seien Sie so genau wie möglich bei der Problembeschreibung, da dies uns dabei hilft, das Problem so schnell wie möglich zu beheben.

```
Dokument-URL:

Nummer und Name des Abschnitts:

Problembeschreibung:

Verbesserungsvorschlag:

Weitere Informationen:
```

Bitte geben Sie auch Ihren Namen an, damit wir uns offiziell bei Ihnen bedanken können.

Kapitel 1. Über JBoss Messaging 1.4

Bei JBoss Messaging handelt es sich um ein neues Enterprise Messaging System von JBoss. Dabei wurde JBossMQ, der vorangehenden JBoss JMS Provider, vollständig umgeschrieben.

JBoss Messaging ist der standardmäßige JMS-Provider in der JBoss Enterprise Application Platform 4.3 und 5.

JBoss Messaging ist ein integraler Bestandteil von Red Hats Messaging Strategie. Es bietet Performance-Verbesserungen in sowohl einzelnen Nodes als auch in geclusterten Umgebungen und bietet eine modulare Architektur, so dass wir auf einfache Weise auch in Zukunft weitere Features hinzufügen können.

Dieses Benutzerhandbuch zeigt Ihnen die Installation, die Einstellungen und Konfiguration von JBoss Messaging für die JBoss Enterprise Application Platform.

Kapitel 2. Einführung

JBoss Messaging bietet eine Open Source und standardbasierte Messaging Platform, das Messaging der Enterprise-Klasse für einen Massenmarkt ermöglicht.

JBoss Messaging implementiert einen High Performance und robusten Messaging Kern, der entworfen wurde, um die größten und am stärksten genutzten Service Oriented Architectures(SOAs), Enterprise Service Buses (ESBs) sowie anderen Integrationsbedarf unabhängig vom Bedarfsniveau zu unterstützen.

JBoss Messaging gestattet die gleichmäßige Verteilung der Applikationslast über Ihren Cluster hinweg. Es balanciert die CPU-Zyklen jedes Nodes ohne Single-Point-of-Failure, wodurch eine äußerst skalierbare und performante Clustering-Implementierung möglich wird.

JBoss Messaging beinhaltet ein Java Messaging Service (JMS) Frontend, so dass Nachrichten in standardbasiertem Format geliefert werden, und in Zukunft Support für andere Nachrichtenprotokolle möglich wird.

2.1. JBoss Messaging Features

JBoss Messaging bietet die folgenden Features:

- Einen starken Fokus auf Performance, Zuverlässigkeit und Skalierbarkeit mit hohem Durchsatz und niedriger Latenz.
- Eine Grundlage für JBoss ESB für SOA Initiativen. (JBoss ESB verwendet JBoss Messaging als seinen standardmäßigen JMS-Provider.)

JBoss Messaging beinhaltet auch:

- "Publish-Subscribe" und "Point-to-Point" Messaging Modelle,
- Persistente und nicht-persistente Nachrichten,
- Garantierte Message-Delivery, dass Nachrichten, wo erforderlich, ein Mal (und nur ein Mal) ausgeliefert werden,
- ein transaktionales und zuverlässiges Interface, das die ACID-Semantik unterstützt,
- ein anpassbares Sicherheits-Framework basierend auf JAAS
- Komplette Integration mit JBoss Transactions (früher bekannt als Arjuna JTA) für vollständige Transaktionswiederherstellbarkeit,
- Umfassende JMX-Management Oberfläche,
- Support für die meisten wichtigen Datenbanken wie Oracle, DB2, Sybase, MS SQL Server, PostgreSQL und MySQL,
- HTTP-Transport, um die Verwendung durch Firewalls zu gestatten, die nur HTTP-Traffic erlauben,
- Servlet-Transport für das Messaging durch ein zugewiesenes Servlet,
- SSL-Transport,
- Konfigurierbare DLQs (Dead Letter Queues) und Expiry Queues,
- Nachrichtenstatistiken: Bieten eine rollende historische Ansicht, welche Nachrichten an welche Warteschlangen und Abonnements geliefert wurden,
- Automatisches Paging von Nachrichten zum Speicher. Gestattet die Verwendung sehr langer Warteschlangen - zu lang, um auf einmal in den Speicher zu passen und
- strenge Nachrichtenreihenfolge, wodurch zu einer bestimmten Nachrichtengruppe gehörende Nachrichten gemäß ihrer Ankunft in der Zielwarteschlange ausgeliefert werden.

JBoss Messaging beinhaltet außerdem die folgenden Clustering Features:

- **Vollständig geclusterte Warteschlangen und Topics**

  Logische Warteschlangen und Topics werden über den Cluster hinweg distribuiert. Sie können von jedem Node aus an eine Warteschlange oder ein Topic senden und von einem anderen empfangen.

- **Vollständig geclusterte dauerhafte Abonnements**

Auf ein bestimmtes dauerhaftes Abonnement kann von jedem Node des Clusters aus zugegriffen werden - wodurch die Verarbeitungslast dieses Abonnements über das Cluster verteilt werden kann.

- **Vollständig geclusterte temporäre Warteschlangen**

  Senden einer Nachricht mit einem **`replyTo`** einer temporären Warteschlange und kann zu jedem beliebigen Node des Clusters zurückgeschickt werden.

- **Intelligente Nachrichten-Redistribuierung**

  Nachrichten werden automatisch zwischen verschiedenen Nodes des Clusters verschoben, wenn Verbraucher auf einem Node schneller als auf einem anderen sind. Dies kann mangelnder Ausnutzung oder aber Überlastung auf bestimmten Nodes vorbeugen.

- **Reihenfolgenschutz von Nachrichten**

  Falls Sie sichergehen wollen, dass die Reihenfolge der von einem Producer produzierten Nachrichten dieselbe dieselbe ist wie vom Verbraucher verbraucht, so aktivieren Sie dies. Dies funktioniert selbst bei aktiver Nachrichten-Redistribuierung.

- **Vollständig transparenter Failover**

  Schlägt der Server fehl, fahren Ihre Sessions ohne Ausnahmeldung an einem neuen Node fort als sei nichts passiert. Vollständig konfigurierbar - falls Sie dies nicht wünschen können Sie auf Ausnahmemeldungen und die manuelle erneute Herstellung von Verbindungen an einem anderen Node zurückgreifen.

- **Hohe Verfügbarkeit und nahtloser Failover**

  Schlägt der Node, mit dem Sie verbunden sind fehl, erfolgt ein automatischer Failover an einen anderen Node und Sie verlieren keine der persistenten Nachrichten. Sie können nahtlos mit Ihrer Session fortfahren, wo Sie aufgehört haben. *Die einmalige (und ausschließlich einmalige)* Auslieferung persistenter Nachrichten wird zu allen Zeiten berücksichtigt.

- **Message-Bridge**

  JBoss Messaging enthält eine Message Bridge Komponente, die es Ihnen gestattet, Nachrichten zwischen zwei beliebigen JMS1.1 Destinations an demselben oder aber an verschiedenen Orten zu überbrücken (z.B. durch ein WAN getrennt). Dies ermöglicht Ihnen die Verbindung mit geografisch getrennten Clustern, die riesige und global verteilte logische Warteschlangen und Topics bilden.

2.2. Kompatibilität mit JBossMQ

JBoss MQ war die mit der Enterprise Application Platform 4.2 gelieferte JMS-Implementierung. Da JBoss Messaging JMS 1.1 und JMS 1.0.2b kompatibel ist, läuft der gegen JBossMQ geschriebene JMS-Code ohne Änderungen bei JBoss Messaging.

JBoss Messaging besitzt keine "Wire"-Format Kompatibilität mit JBossMQ daher wäre es notwendig JBoss MQ Clients mit JBoss Messaging Client Jars zu upgraden.

Wichtig

Selbst wenn JBoss Messaging Deployment-Deskriptoren den JBoss MQ Deployment-Deskriptoren sehr ähnlich sind, sind diese *nicht identisch*, so dass einige einfache Anpassungen notwendig sind, damit sie mit JBoss Messaging funktionieren. Auch das Datenmodell der Datenbank ist ganz anders, versuchen Sie also nicht, JBoss Messaging mit einem JBoss MQ Datenschema und umgekehrt zu benutzen.

2.3. Von JBoss Messaging verwendete System-Properties

2.3.1. support.bytesId

Diese System-Property steuert das Standardverhalten wenn ein **`JBossMessage`**-Objekt aus einem fremden Nachrichtenobjekt konstruiert wird. Stellen Sie diese Property beim Server-Startup mit der **`-D`**-Option mittels Befehlszeile ein.

Ist diese Property auf **`true`** eingestellt, so versucht der **`JBossMessage`**-Konstruktor die native byte[] Korrelations-ID aus den fremden Nachrichten-Headern zu extrahieren. Ist sie auf **`false`** eingestellt, so wird der normale String-Typ **`JMSCorrelationID`** verwendet. Die Standardeinstellung dieser Property ist **`true`**, falls sie nicht eingestellt oder aber auf etwas anderes als **`true`** oder **`false`** eingestellt ist.

2.3.2. retain.oldxabehaviour

Diese System-Property steuert die Art von Ausnahme, die von einer JMS XAResource gemeldet wird, falls die **`prepare()`**-Methode aufgerufen wird, nachdem die Verbindung unterbrochen wurde. Stellen Sie diese Property beim Server-Startup mit der **`-D`**-Option mittels Befehlszeile ein.

Ist diese Property nicht definiert, so wird eine **`XAException`** mit einem **`XA_RBCOMMFAIL`**-Fehlercode gemeldet. Andernfalls wird eine **`XAException`** mit einem **`XA_RETRY`**-Fehlercode gemeldet. Bitte beachten Sie, dass JBoss Messaging diese Property nicht standardmäßig definiert.

Kapitel 3. JBoss Messaging Installation

JBoss Enterprise Application Platform (EAP) wird mit JBoss Messaging als vorinstalliertem, standardmäßigem JMS-Provider geliefert. Falls Sie EAP Version 4.3 oder höher verwenden, so besteht keine Notwendigkeit JBoss Messaging manuell zu installieren.

Kapitel 4. Ausführen der Beispiele

JBoss Messaging bietet eine Reihe von Beispielen, die zum Download bereitstehen. Im mit der JBoss Enterprise Application Platform vertriebenen *Installation Guide* finden Sie Informationen zur Installation und Konfiguration der Beispiele. Der *Installation Guide* ist unter http://docs.redhat.com/ unter *JBoss Enterprise Application Platform > 5.0* verfügbar.

Entzippen Sie die Beispiele ins **$JBOSS_HOME/docs/examples**-Verzeichnis.

Ehe Sie diese Beispiele ausführen, deployen Sie den **jbm-examples-destinations-service** unter **$JBOSS_HOME/docs/examples/jboss-messaging-examples/destinations/** nach **$JBOSS_HOME/server/default/deploy**.

Die folgenden Beispiele sind in JBoss Messaging enthalten:

$JBOSS_HOME/docs/examples/jboss-messaging-examples/queue/

Dieses Beispiel demonstriert ein einfaches 'send' (senden) und 'receive' (empfangen) an eine Remote-Warteschlange mittels eines JMS-Client.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/queue-failover/

Dieses Beispiel demonstriert den transparenten Failover eines JMS-Verbrauchers.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/topic/

Dieses Beispiel demonstriert ein einfaches 'send' (senden) und 'receive' (empfangen) an ein Remote-Topic mittels eines JMS-Client.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/mdb/

Dieses Beispiel demonstriert den Gebrauch eines Enterprise JavaBean 2.1 MDB mit JBoss Messaging.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/ejb3mdb/

Dieses Beispiel demonstriert den Gebrauch eines Enterprise JavaBean 3.0 MDB mit JBoss Messaging.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/stateless/

Dieses Beispiel demonstriert ein Enterprise JavaBean 2.1 "Stateless Session Bean", das mit JBoss Messaging interagiert.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/mdb-failure/

Dieses Beispiel demonstriert Zurücksetzen und erneutes Ausliefern mit einem Enterprise JavaBean 2.1.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/secure-socket/

Dieses Beispiel demonstriert die Interaktion eines JMS Client mit einem JBoss Messaging Server unter Verwendung von SSL verschlüsseltem Transport.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/http/

Dieses Beispiel demonstriert die Interaktion eines JMS Client mit einem JBoss Messaging Server und Tunneling von Traffic über HTTP-Protokoll.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/web-service/

Dieses Beispiel demonstriert die Interaktion des JBoss Web-Dienstes mit JBoss Messaging.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/distributed-queue/

Dieses Beispiel demonstriert die Interaktion eines JMS Clients mit einer JBoss Messaging distribuierten Warteschlange - zur Ausführung werden zwei JBoss Application Plattform Instanzen benötigt.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/distributed-topic/

Dieses Beispiel demonstriert die Interaktion eines JMS Clients mit einem JBoss Messaging distribuierten Topic - zur Ausführung werden zwei JBoss Enterprise Application Platform Instanzen benötigt.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/servlet/

Dieses Beispiel demonstriert die Verwendung des Servlet Transports mit JBoss Messaging. Es deployt ein Servlet und eine **ConnectionFactory**, die den Servlet-Transport verwendet.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/ordering-group/

Dieses Beispiel demonstriert die Verwendung strenger Nachrichtenreihenfolge. Es verwendet das JBoss Messaging "Ordering Group"-API zur Lieferung streng geordneter Nachrichten, unabhängig von der Nachrichtenpriorität.

$JBOSS_HOME/docs/examples/jboss-messaging-examples/bridge/

Dieses Beispiel demonstriert die Anwendung einer Nachrichtenbrücke (Message Bridge). Es deployt eine Nachrichtenbrücke in der JBoss Enterprise Application Platform, die dann Nachrichten von einer Quelle zu einer Zielwarteschlange verschiebt.

Machen Sie sich mit den Beispielen vertraut, um ein besseres Verständnis von JBoss Messaging zu gewinnen.

Wichtig - Nicht geclusterte Beispiele

Die nicht geclusterten Beispiele erwarten eine mit allen Standardeinstellungen laufende JBoss Enterprise Application Platform. Die **readme.html**-Datei bietet Setup-Informationen, zu erwartende Ausgaben und einfache Problembehebungen.

Wichtig - geclusterte Beispiele

Die geclusterten Beispiele erfordern, dass zwei JBoss Enterprise Application Platform mit korrekt konfigurierten Cluster-Namen und Port-Einstellungen laufen. Die **readme.html**-Datei bietet Setup-Informationen, zu erwartende Ausgaben und einfache Problembehebungen.

Kapitel 5. Konfiguration

Das Java Message Service (JMS) API legt fest, wie ein Messaging Client mit einem Messaging Server interagiert. Wie Messaging Dienste, wie etwa Message-Destinationen und Connection Factories definiert und implementiert werden, hängt vom JMS-Provider ab. JBoss Messaging hat seine eigenen Dateien zur Dienstkonfiguration.

Dieses Kapitel zeigt Ihnen, wie Sie die verschiedenen, in JBoss Messaging verfügbaren Dienste konfigurieren, die zusammenarbeiten, um JMS Dienste auf API-Ebene für Client Applikationen bereitstellen.

Die Konfiguration des JBoss Messaging Dienstes verteilt sich auf mehrere Konfigurationsdateien. Je nach dem Typ des bereitgestellten Dienstes werden Konfigurationsinformationen zwischen **`messaging-service.xml`**, **`remoting-bisocket-service.xml`**, **`<your database type>-persistence-service.xml`**, **`connection-factories-service.xml`** und **`destinations-service.xml`** geteilt. Sie finden alle diese Dateien im **`$JBOSS_HOME/server/$PROFILE/deploy/messaging`**-Verzeichnis.

Die AOP Interceptor-Stacks werden in **`aop-messaging-client.xml`** (für Client-Seiten Verhalten) and **`aop-messaging-server.xml`** (für Server-Seiten Verhalten) konfiguriert. Normalerweise werden Sie diese nicht ändern müssen, sondern es können einige Interceptoren entfernt werden, um die Performance etwas zu verbessern (falls Sie diese nicht brauchen). Seien Sie vorsichtig und stellen Sie sicher, dass Sie die Sicherheitskonsequenzen berücksichtigt haben, ehe Sie den Security-Interceptor entfernen.

5.1. Konfiguration von ServerPeer

Der **`ServerPeer`** ist das Herz der JBoss Messaging JMS Fassade. Sie können dessen Verhalten durch Bearbeitung von **`$JBOSS_HOME/server/$PROFILE/deploy/messaging/messaging-service.xml`** konfigurieren.

All JBoss Messaging Dienste basieren im **`ServerPeer`**.

Unten sehen Sie ein Beispiel für eine Server Peer Konfiguration. Bitte beachten Sie, dass nicht alle Werte für die Attribute des Server Peer in dem Beispiel spezifiziert sind

```
<!-- ServerPeer MBean configuration
     ============================== -->

<mbean code="org.jboss.jms.server.ServerPeer"
  name="jboss.messaging:service=ServerPeer"
  xmbean-dd="xmdesc/ServerPeer-xmbean.xml">

  <!--The unique id of the server peer - in a cluster each node
      MUST have a unique value - must be an integer-->

  <attribute name="ServerPeerID">
    ${jboss.messaging.ServerPeerID:0}
  </attribute>

  <!--The default JNDI context to use for queues when they are
      deployed without specifying one-->

  <attribute name="DefaultQueueJNDIContext">/queue</attribute>

  <!--The default JNDI context to use for topics when they are
      deployed without specifying one -->

  <attribute name="DefaultTopicJNDIContext">/topic</attribute>

  <attribute name="PostOffice">
    jboss.messaging:service=PostOffice
  </attribute>

  <!-- The default Dead Letter Queue (DLQ) to use for destinations.
       This can be overridden on a per destinatin basis -->

  <attribute name="DefaultDLQ">
    jboss.messaging.destination:service=Queue,name=DLQ
  </attribute>

  <!--The default maximum number of times to attempt delivery of a
      message before sending to the DLQ (if configured).
      This can be overridden on a per destination basis-->

  <attribute name="DefaultMaxDeliveryAttempts">10</attribute>

  <!--The default Expiry Queue to use for destinations. This can
      be overridden on a per destinatin basis-->

  <attribute name="DefaultExpiryQueue">
    jboss.messaging.destination:service=Queue,name=ExpiryQueue
  </attribute>

  <!--The default redelivery delay to impose. This can be overridden
      on a per destination basis -->

  <attribute name="DefaultRedeliveryDelay">0</attribute>

  <!--The periodicity of the message counter manager enquiring on
      queues for statistics-->

  <attribute name="MessageCounterSamplePeriod">5000</attribute>

  <!--The maximum amount of time for a client to wait for failover
      to start on the server side after it has detected failure-->

  <attribute name="FailoverStartTimeout">60000</attribute>

  <!--The maximum amount of time for a client to wait for failover
      to complete on the server side after it has detected failure-->

  <attribute name="FailoverCompleteTimeout">300000</attribute>
```

```
  <attribute name="StrictTck">false</attribute>

  <!--The maximum number of days results to maintain in the message
      counter history-->

  <attribute name="DefaultMessageCounterHistoryDayLimit">-1</attribute>

  <!--The name of the connection factory to use for creating
      connections between nodes to pull messages-->

  <attribute name="ClusterPullConnectionFactoryName">
    jboss.messaging.connectionfactory:service=ClusterPullConnectionFactory
  </attribute>

  <!--When redistributing messages in the cluster. Do we need to
      preserve the order of messages received
      by a particular consumer from a particular producer? -->

  <attribute name="DefaultPreserveOrdering">false</attribute>

  <!-- Max. time to hold previously delivered messages back waiting for
       clients to reconnect after failover -->

  <attribute name="RecoverDeliveriesTimeout">300000</attribute>

  <!-- Set to true to enable message counters that can be viewed via JMX -->

  <attribute name="EnableMessageCounters">false</attribute>

  <!-- The password used by the message sucker connections to create connections.
       THIS SHOULD ALWAYS BE CHANGED AT INSTALL TIME TO SECURE SYSTEM
  <attribute name="SuckerPassword"></attribute>
  -->

  <!-- The name of the server aspects configuration resource
  <attribute name="ServerAopConfig">aop/jboss-aop-messaging-server.xml</attribute>
  -->
  <!-- The name of the client aspects configuration resource
    <attribute name="ClientAopConfig">aop/jboss-aop-messaging-
client.xml</attribute>
  -->

  <depends optional-attribute-name="PersistenceManager">
    jboss.messaging:service=PersistenceManager
  </depends>

  <depends optional-attribute-name="JMSUserManager">
    jboss.messaging:service=JMSUserManager
  </depends>

  <depends>jboss.messaging:service=Connector,transport=bisocket</depends>
  <depends optional-attribute-name="SecurityStore"
    proxy-type="org.jboss.jms.server.SecurityStore">
      jboss.messaging:service=SecurityStore
  </depends>
</mbean>
```

5.2. ServerPeer-Attribute

Dieser Abschnitt erläutert die **`ServerPeer`**-gemanagten Bean-Attribute.

ServerPeerID

Der eindeutige Bezeichner des **`ServerPeer`**. Jeder durch Sie deployte Node muss einen eindeutigen Bezeichner besitzen. Dies gilt für verschiedene Nodes, die ein Cluster bilden wie auch für Nodes die lediglich über eine Nachrichtenbrücke verbunden sind. Bei dem Bezeichner muss es sich um eine gültige ganze Zahl handeln.

DefaultQueueJNDIContext

Der beim Binding von Warteschlangen zu verwendende standardmäßige JNDI-Kontext. Der Standardwert ist **`/queue`**.

DefaultTopicJNDIContext

Der beim Binding von Topics zu verwendende standardmäßige JNDI-Kontext. Der Standardwert ist **`/topic`**.

PostOffice

Dies ist das Post Office das der **`ServerPeer`** verwendet. In der Regel müssen Sie dieses Attribut nicht verändern. Das Post Office ist verantwortlich für das Routing von Nachrichten an Warteschlangen und die Wartung des Mapping zwischen Adressen und Warteschlangen.

DefaultDLQ

Dies ist der Name der standardmäßigen DLQ (Dead Letter Queue), die der Server für Destinationen verwendet. Die DLQ kann auf pro-Destinationsbasis außer Kraft gesetzt werden - weitere Informationen finden Sie unter Abschnitt 5.7, „Konfiguration von Destinationen“. Eine DLQ ist eine spezielle Destination, wo Nachrichten gesendet werden, wenn der Server mehr als eine bestimmte Anzahl von Malen erfolglos versucht hat, diese auszuliefern. Ist die DLQ überhaupt nicht festgelegt, so wird die Nachricht nach der maximalen Anzahl von Auslieferungsversuchen entfernt. Die maximale Anzahl von Auslieferungsversuchen kann mittels des Attributes **`DefaultMaxDeliveryAttempts`** für einen allgemeingültigen Standard oder aber individuell auf per-Destinationsbasis festgelegt werden.

DefaultMaxDeliveryAttempts

Die maximale Anzahl von Malen, die der Versuch der Auslieferung einer Nachricht unternommen wird, ehe die Nachricht an die DLQ geschickt wird (falls konfiguriert). Der Standardwert ist **`10`**. Sie können diesen Wert auf per-Destinationsbasis außer Kraft setzen.

DefaultExpiryQueue

Dies ist der Name der Standard Expiry-Warteschlange (Ablauf-Warteschlange), die der **`ServerPeer`** für Destinationen verwendet. Der Ablauf kann auf per-Destinationsbasis außer Kraft gesetzt werden - Einzelheiten entnehmen Sie bitte der MBean-Konfiguration für Destinationen. Bei einer Expiry-Warteschlange handelt es sich um eine besondere Destination, bei der Nachrichten gesendet werden, wenn sie abgelaufen sind. Message-Expiry (Nachrichtenablauf) wird durch den Wert von **`Message::getJMSExpiration()`** bestimmt. Ist die Expiry-Warteschlange überhaupt nicht festgelegt, so wird die Nachricht nach ihrem Ablauf entfernt.

DefaultRedeliveryDelay

Dieses Attribut gestattet die Verzögerung eines erneuten Auslieferungsversuchs, was Ihnen bei der Vermeidung einer Thrashing Delivery-Failure hilft. Der Standardwert lautet **`0`** (d.h. keine Verzögerung). Sie können diesen Wert auf per-Destination Basis außer Kraft setzen.

MessageCounterSamplePeriod

Dieses Attribut definiert den Zeitraum zwischen den Anfragen des Servers bis zur Warteschlange für Warteschlangenstatistiken. Der Standardwert sind **`5000`** Millisekunden.

FailoverStartTimeout

Die maximale Anzahl von Millisekunden, die der Client auf Failover wartet, um auf der Serverseite zu starten, wenn ein Problem auftritt. Der Standardwert ist **`60000`** (eine Minute).

FailoverCompleteTimeout

Die maximale Anzahl von Millisekunden, die der Client auf Failover wartet, um auf der Serverseite zu beenden, nachdem er gestartet hat. Der Standardwert ist **`300000`** (fünf Minuten).

DefaultMessageCounterHistoryDayLimit

JBoss Messaging bietet eine Nachrichtenzähler-Historie, die die Anzahl von in jeder Warteschlange ankommenden Nachrichten für eine bestimmte Anzahl von Tagen anzeigt. Dieses Attribut repräsentiert die maximale Anzahl von Tagen, die die Nachrichtenzähler-Historie gespeichert wird. Sie kann au auf pro-Destinationsbasis außer Kraft gesetzt werden.

ClusterPullConnectionFactoryName

Die Connection Factory mit der Nachrichten zwischen Warteschlangen gezogen oder "gesaugt" werden. Sie können dieses Attribut weglassen, um das Ziehen/"Saugen" von Nachrichten zu deaktivieren, während der Failover erhalten bleibt.

DefaultPreserveOrdering

Falls **`true`**, so wird strenge JMS-Reihenfolge im Cluster bewahrt. Einzelheiten finden Sie unter Kapitel 6, *Clustering Hinweise*. Der Standard lautet **`false`**.

RecoverDeliveriesTimeout

Tritt ein Failover auf, so werden bereits ausgelieferte Nachrichten beiseite gehalten während darauf gewartet wird, dass Clients die Verbindung wieder herstellen. Falls die Clients sich überhaupt nicht mehr verbinden (z.B. der Client ist tot), so erfolgt nach und nach ein Timeout bei den Nachrichten und diese werden diese werden der Warteschlange wieder hinzugefügt. Dieses Attribut setzt den Zeitrahmen vor dem Timeout in Millisekunden. Der Standardwert ist **`300000`** (fünf Minuten).

EnableMessageCounters

Zur Aktivierung der Nachrichtenzähler auf **`true`** setzen, wenn der Server startet.

SuckerPassword

JBoss Messaging stellt intern Verbindungen zwischen Knoten her, um Nachrichten zwischen geclusterten Destinationen zu redistribuieren. Diese Verbindungen werden mit einem speziellen, reservierten Benutzernamen hergestellt. Dieses Attribut definiert das beim Erstellen dieser Verbindungen zu verwendende Passwort.

Für spätere Versionen von JBoss Messaging als 1.4.1.GA müssen Sie das **`SuckerPassword`** im **`SecurityMetadataStore`** definieren.

Warnung

Das **`SuckerPassword`** muss zum Installationszeitpunkt festgelegt werden oder es wird das Standardpasswort verwendet. Jeder, der das Standardpasswort kennt, kann auf beliebige Ziele auf dem Server zugreifen. Dieser Wert muss zum Installationszeitpunkt geändert werden.

SuckerConnectionRetryTimes

Die maximale Anzahl von Malen, die eine Sucker-Connection den Versuch wiederholen darf, falls dieser fehlschlägt. Der Standardwert ist **`-1`**, welcher für "retry indefinitely" (Versuch undendlich wiederholen) steht.

SuckerConnectionRetryInterval

Dies ist der Intervall in Millisekunden zwischen den erneuten Versuchen der fehlgeschlagenen Sucker-Connection. Der Standardwert lautet **`5000`**.

StrictTck

Um strikte JMS Technology Compatibility Kit (TCK) Semantik zu aktivieren, setzen Sie dieses Attribut auf **`true`**.

Ziele

Gibt eine Liste aktuell deployter Destinationen (Warteschlangen und Topics) wieder.

MessageCounters

Ein Nachrichtenzähler für eine bestimmte Warteschlange.

MessageStatistics

Statistiken für jeden Nachrichtenzähler für jede Warteschlange.

SupportsFailover

Ist dieses Attribut auf **`false`** eingestellt, so findet der Failover auf Serverseite nicht statt, wenn ein Knoten in einem Cluster abstürzt.

PersistenceManager

Dies ist der Persistenz-Manager den der **`ServerPeer`** verwendet. Sie werden dieses Attribut in der Regel nicht ändern müssen.

JMSUserManager

Dies ist der JMS Benutzer-Manager den der **`ServerPeer`** verwendet. Sie werden dieses Attribut in der Regel nicht ändern müssen.

SecurityStore

Der einbindbare **`SecurityStore`**. Falls Sie dieses Attribut neu definieren, vergessen Sie nicht, dass Sie den **`MessageSucker`**-Benutzer (**`JBM.SUCKER`**) mit allen von Clustering benötigten Sondergenehmigungen authentifizieren müssen.

SupportsTxAge

Legt fest, ob die Transaktionserstellungszeit im Transaktionsspeicher gespeichert wird. Falls auf **`true`** gesetzt, so wird diese gespeichert. Der Standardwert lautet **`false`**.

5.2.1. ServerPeer Methoden

Die folgenden Methoden sind für das ServerPeer gemanagte Bean verfügbar:

deployQueue

Wird für das programmatische Deployment einer Warteschlange verwendet. Falls die Warteschlange bereits existiert, aber noch undeployt ist, so wird diese deployed. Andernfalls wird sie erstellt und deployt.

Der **`name`**-Parameter stimmt mit der zu deployenden Destination überein.

Der optionale **`jndiName`**-Parameter repräsentiert den vollständigen jndi-Namen, wo die Destination gebunden werden soll. Ist dies nicht festgelegt, so wird die Destination in **`<DefaultQueueJNDIContext>/<name>`** gebunden.

Es existieren zwei überladene Versionen dieses Vorgangs. Die erste Version dieser Operation deployt die Destination mit den standardmäßigen Paging-Parametern. Die zweite Version deployt die Destination mit den festgelegten Paging-Parametern. Weitere Informationen zu Paging-Parametern finden Sie unter Abschnitt 5.7, „Konfiguration von Destinationen“.

undeployQueue

Wird für das programmatische Undeployment einer Warteschlange verwendet. Diese Operation gibt **`true`** wieder, wenn die Warteschlange erfolgreich undeployt wurde, andernfalls wird **`false`** wiedergegeben.

destroyQueue

Wird für die programmatische Löschung einer Warteschlange verwendet. Warteschlangen werden undeployt, und dann werden alle ihre Daten in der Datenbank gelöscht.

Warnung

Seien Sie vorsichtig bei der Verwendung dieser Methode, da sie sämtliche Daten für die Warteschlange löscht.

Diese Operation antwortet mit **`true`**, falls die Warteschlange erfolgreich gelöscht wurde. Ist dies nicht der Fall, so liefert sie **`false`**.

deployTopic

Wird für das programmatische Deployment eines Topics verwendet. Es existieren zwei überladene Versionen dieses Vorgangs. Die erste Version dieser Operation deployt bereits bestehende Topics mit den standardmäßigen Paging-Parametern. Die zweite erstellt und deployt Topics mit festgelegten Paging-Parametern. Weitere Informationen finden Sie unter Abschnitt 5.7, „Konfiguration von Destinationen“.

Der **`name`**-Parameter repräsentiert den Namen der zu deployenden Destination.

Der **`jndiName`**-Parameter repräsentiert den vollständigen JNDI-Namen, wo die Destination gebunden werden soll. Ist dies nicht festgelegt, so wird die Destination in **`<DefaultTopicJNDIContext>/<name>`** gebunden.

undeployTopic

Wird für das programmatische Undeployment eines Topics verwendet. Topics werden undeployt, jedoch nicht aus dem persistenten Speicher entfernt. Diese Operation gibt **`true`** wieder, wenn das Topic erfolgreich undeployt wurde, andernfalls wird **`false`** wiedergegeben.

destroyTopic

Wird für die programmatische Löschung eines Topic verwendet. Topics werden undeployt, und dann werden alle ihre Daten aus der Datenbank entfernt und gelöscht. Diese Operation antwortet mit **`true`**, falls das Topic erfolgreich gelöscht wurde. Ist dies nicht der Fall, so liefert sie **`false`**.

Warnung

Seien Sie bei der Verwendung dieser Methode vorsichtig, da sie sämtliche Daten für das Topic löscht.

listMessageCountersHTML

Liefert Nachrichtenzähler in einfach anzuzeigendem HTML-Format.

resetAllMesageCounters

Setzt den Nachrichtenzähler auf Null zurück.

enableMessageCounters

Aktiviert alle Nachrichtenzähler (Message Counters) für alle Destinationen. Nachrichtenzähler sind standardmäßig deaktiviert.

disableMessageCounters

Deaktiviert alle Nachrichtenzähler (Message Counters) für alle Destinationen. Nachrichtenzähler sind standardmäßig deaktiviert.

retrievePreparedTransactions

Ruft eine Liste aller Xids für alle Transaktionen ab, die zum aktuellen Zeitpunkt in einem vorbereiteten Status am Node vorhanden sind.

showPreparedTransactions

Ruft eine in einfachem HTML-Format darstellbare Liste aller Xids für alle Transaktionen ab, die zum aktuellen Zeitpunkt in einem vorbereiteten Status am Knoten vorhanden sind.

listAllPreparedTransactions

Zeigt Einzelheiten zu allen vorbereiteten Transaktionen an.

listPreparedTransactions

Zeigt Details aller vorbereiteten Transaktionen an, wo die Transaktionsalter älter oder gleich einer bestimmten Zeit sind.

showMessageDetails

Zeigt die Einzelheiten einer Nachricht an. Die Message-ID wird zur Bestimmung der anzuzeigenden Nachricht verwendet.

commitPreparedTransaction

Manuelle Festschreibung einer vorbereiteten Transaktion. Die Transaktions-ID wird zur Bestimmung der festzuschreibenden Transaktion verwendet.

rollbackPreparedTransaction

Manuelle Zurücksetzung einer vorbereiteten Transaktion. Die Transaktions-ID wird zur Bestimmung der zurückzusetzenden Transaktion verwendet.

5.3. Die Datenbank wechseln

Sie müssen Ihre Datenbank wechseln

Die Standardpersistenzkonfiguration funktioniert mit Hypersonic (HSQLDB), so dass die JBoss Enterprise Plattformen sofort einsatzbereit sind. Jedoch wird *Hypersonic nicht für die Produktion unterstützt und sollte nicht in einer Produktionsumgebung verwendet werden.*
Bekannte Probleme mit der Hypersonic Datenbank sind:

- keine Transaktionsisolierung
- Thread- und Socket-Lecks (**`connection.close()`** bereinigt Ressourcen nicht)
- Persistenzqualität (Protokolle werden nach dem Fehlschlagen häufig korrumpiert, wodurch die automatische Wiederherstellung verhindert wird)
- Datenbankkorruption
- Stabilität bei Belastung (Datenbankprozesse stellen den Betrieb bei zu vielen Daten ein)
- nicht funktionsfähig in geclusterten Umgebungen

Im "Using Other Databases"-Kapitel des *Getting Started Guide* finden Sie weitere Informationen.

Persistence Manager, Post Office und JMS User Manager interagieren alle mit persistentem Speicher. Der Persistence Manager handhabt nachrichtenbezogene Persistenz. Das Post Office handhabt bindungsbezogene Persistenz. Der JMS User Manager handhabt benutzerbezogene Persistenz. Sämtliche Konfiguration für diese gemanagten Beans wird in der **`<your database type>-persistence-service.xml`**-Datei gehandhabt.

Beispiel-Konfigurationsdateien für MySQL, Oracle, PostgreSQL, Microsoft SQL Server oder Sybase Datenbanken sind im **`jboss-as/docs/examples/jms`**-Verzeichnis des Release-Bündels verfügbar.

Um den Support für eine dieser Datenbanken zu aktivieren, ersetzen Sie die standardmäßige **`jboss-as/server/$PROFILE/deploy/messaging/hsqldb-persistence-service.xml`**-Konfigurationsdatei mit der für Ihren Datenbanktyp spezifischen Jonfigurationsdatei und starten Sie den Server neu.

Von einem Datenspeicher abhängige Messaging Dienste referenzieren **`java:/DefaultDS`** für die Datenquelle. Falls Sie eine Datenquelle mit einem anderen JNDI-Namen deployen, so müssen Sie alle **`DataSource`**-Attribute in der Persistenz-Konfigurationsdatei aktualisieren. Die Distribution bietet Beispielkonfigurationen von Datenquellen.

5.4. Konfiguration des Post Office

Das Post Office routet Nachrichten an deren Destination oder Destinationen. Es wartet die Mappings zwischen den Adressen an die die Nachricht geschickt werden kann und der endgültigen Warteschlange. Beim Senden einer Nachricht mit einer die JMS-Warteschlange repräsentierenden Adresse etwa, routet das Post Office die Nachricht an diese JMS-Warteschlange. Beim Senden einer Nachricht mit einer das JMS-Topic repräsentierenden Adresse, routet das Post Office die Nachricht an einen Satz von Warteschlangen — eine für jedes JMS-Abonnement.

Das Post Office handhabt auch die Persistenz für das Mapping von Adressen.

JBoss Messaging Post-Offices sind außerdem Cluster gewahr. In einem Cluster werden sie automatisch geroutet und ziehen untereinander Nachrichten zwischen Knoten, um vollständig distribuierte Warteschlangen und Topics liefern zu können.

Konfigurieren Sie das Post Office in der **`<database type>-persistence-service.xml`**-Datei. Zum Beispiel:

```
<mbean code="org.jboss.messaging.core.jmx.MessagingPostOfficeService"
  name="jboss.messaging:service=PostOffice"
  xmbean-dd="xmdesc/MessagingPostOffice-xmbean.xml">

  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>

  <depends>
    jboss.jca:service=DataSourceBinding,name=DefaultDS
  </depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The name of the post office -->

  <attribute name="PostOfficeName">JMS post office</attribute>

  <!-- The datasource used by the post office to access it's
       binding information -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on
       every start-up -->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then we will automatically detect and reject
       duplicate messages sent during failover -->

  <attribute name="DetectDuplicates">true</attribute>

  <!-- The size of the id cache to use when detecting duplicate
       messages -->

  <attribute name="IDCacheSize">500</attribute>

  <attribute name="SqlProperties">
    CREATE_POSTOFFICE_TABLE=CREATE TABLE JBM_POSTOFFICE
      (POSTOFFICE_NAME VARCHAR(255),
      NODE_ID INTEGER, QUEUE_NAME VARCHAR(255), COND VARCHAR(1023),
      SELECTOR VARCHAR(1023), CHANNEL_ID BIGINT, CLUSTERED CHAR(1),
      ALL_NODES CHAR(1),
      PRIMARY KEY(POSTOFFICE_NAME, NODE_ID, QUEUE_NAME)) ENGINE = INNODB

    INSERT_BINDING=INSERT INTO JBM_POSTOFFICE
      (POSTOFFICE_NAME, NODE_ID, QUEUE_NAME, COND, SELECTOR,
       CHANNEL_ID, CLUSTERED, ALL_NODES)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)

    DELETE_BINDING=DELETE FROM JBM_POSTOFFICE WHERE
      POSTOFFICE_NAME=? AND NODE_ID=? AND QUEUE_NAME=?

    LOAD_BINDINGS=SELECT QUEUE_NAME, COND, SELECTOR,
      CHANNEL_ID, CLUSTERED, ALL_NODES FROM
   JBM_POSTOFFICE WHERE POSTOFFICE_NAME=? AND NODE_ID=?
  </attribute>

  <!-- This post office is clustered. If you don't want a clustered post
       office then set to false -->

  <attribute name="Clustered">true</attribute>

  <!-- All the remaining properties only have to be specified if the post
       office is clustered. You can safely comment them out if your post
```

```
      office is non clustered -->

  <!-- The JGroups group name that the post office will use -->

  <attribute name="GroupName">
    ${jboss.messaging.groupname:MessagingPostOffice}
  </attribute>

  <!-- Max time to wait for state to arrive when the post office
       joins the cluster -->

  <attribute name="StateTimeout">5000</attribute>

  <!-- Max time to wait for a synchronous call to node members using
       the MessageDispatcher -->

  <attribute name="CastTimeout">50000</attribute>

  <!-- Set this to true if you want failover of connections to occur
       when a node is shut down -->

  <attribute name="FailoverOnNodeLeave">false</attribute>

  <!-- JGroups stack configuration for the data channel - used for sending
       data across the cluster -->

  <!-- By default we use the TCP stack for data -->
  <attribute name="DataChannelConfig">
    <config>
      <TCP start_port="7900"
        loopback="true"
        recv_buf_size="20000000"
        send_buf_size="640000"
        discard_incompatible_packets="true"
        max_bundle_size="64000"
        max_bundle_timeout="30"
        use_incoming_packet_handler="true"
        use_outgoing_packet_handler="false"
        down_thread="false" up_thread="false"
        enable_bundling="false"
        use_send_queues="false"
        sock_conn_timeout="300"
        skip_suspected_members="true"/>
      <MPING timeout="4000"
        bind_to_all_interfaces="true"
        mcast_addr="${jboss.messaging.datachanneludpaddress:228.6.6.6}"
        mcast_port="${jboss.messaging.datachanneludpport:45567}"
        ip_ttl="8"
        num_initial_members="2"
        num_ping_requests="1"/>
      <MERGE2 max_interval="100000"
        down_thread="false" up_thread="false" min_interval="20000"/>
      <FD_SOCK down_thread="false" up_thread="false"/>
      <VERIFY_SUSPECT timeout="1500" down_thread="false"
       up_thread="false"/>
      <pbcast.NAKACK max_xmit_size="60000"
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
      <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
      <pbcast.GMS print_local_addr="true" join_timeout="3000"
        down_thread="false" up_thread="false"
        join_retry_timeout="2000" shun="false"
        view_bundling="true"/>
    </config>
```

```
  </attribute>

  <!-- JGroups stack configuration to use for
       the control channel - used for control messages -->

  <!-- We use udp stack for the control channel -->
  <attribute name="ControlChannelConfig">
    <config>
      <UDP
        mcast_addr="${jboss.messaging.controlchanneludpaddress:228.7.7.7}"
        mcast_port="${jboss.messaging.controlchanneludpport:45568}"
        tos="8"
        ucast_recv_buf_size="20000000"
        ucast_send_buf_size="640000"
        mcast_recv_buf_size="25000000"
        mcast_send_buf_size="640000"
        loopback="false"
        discard_incompatible_packets="true"
        max_bundle_size="64000"
        max_bundle_timeout="30"
        use_incoming_packet_handler="true"
        use_outgoing_packet_handler="false"
        ip_ttl="2"
        down_thread="false" up_thread="false"
        enable_bundling="false"/>
      <PING timeout="2000"
        down_thread="false" up_thread="false" num_initial_members="3"/>
      <MERGE2 max_interval="100000"
        down_thread="false" up_thread="false" min_interval="20000"/>
      <FD_SOCK down_thread="false" up_thread="false"/>
      <FD timeout="10000" max_tries="5" down_thread="false"
        up_thread="false" shun="true"/>
      <VERIFY_SUSPECT timeout="1500" down_thread="false"
       up_thread="false"/>
      <pbcast.NAKACK max_xmit_size="60000"
        use_mcast_xmit="false" gc_lag="0"
        retransmit_timeout="300,600,1200,2400,4800"
        down_thread="false" up_thread="false"
        discard_delivered_msgs="true"/>
      <UNICAST timeout="300,600,1200,2400,3600"
        down_thread="false" up_thread="false"/>
      <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
        down_thread="false" up_thread="false"
        max_bytes="400000"/>
      <pbcast.GMS print_local_addr="true" join_timeout="3000"
        use_flush="true" flush_timeout="3000"
        down_thread="false" up_thread="false"
        join_retry_timeout="2000" shun="false"
        view_bundling="true"/>
      <FRAG2 frag_size="60000" down_thread="false" up_thread="false"/>
      <pbcast.STATE_TRANSFER down_thread="false" up_thread="false"
        use_flush="true" flush_timeout="3000"/>
      <pbcast.FLUSH down_thread="false" up_thread="false" timeout="20000"
        auto_flush_conf="false"/>
    </config>
  </attribute>

</mbean>
```

5.4.1. PostOffice-Attribute

Die folgenden Attribute stehen für die Konfiguration des Messaging Post Office zur Verfügung:

DataSource

Legt die Datenquelle fest, die das Post Office zur Persisterung dieser Mapping-Daten verwenden sollte.

SQLProperties

Legt DDL und DML für die bestimmte Datenbank fest. Wird dieses Attribut nicht außer Kraft gesetzt, so wird die standardmäßige Hypersonic Konfiguration für diese Anweisung verwendet.

Wichtig

Hypersonic wird für Produktionsumgebungen nicht unterstützt. Für den Produktionsgebrauch sollte dieser Wert geändert werden.

CreateTablesOnStartup

Legt fest, ob Tabellen und Index bei Start des Post Office erstellt werden. Setzen Sie dies auf **`true`**, wenn Sie möchten, dass das Post Office beim Start die Tabellen (und Indizes) zu erstellen versucht. Falls die Tabellen (oder Indizes) bereits existieren, so wird eine **`SQLException`** durch den JDBC-Treiber gemeldet und vom Persistence Manager ignoriert, so dass fortgefahren werden kann. Der Standardwert lautet **`true`**.

DetectDuplicates

Legt fest, ob das Post Office doppelt versendete Nachrichten aufspürt, wenn die Lieferung an einen neuen Knoten nach Serverausfall eingeschränkt ist. Bei Einstellung auf **`true`**, spürt das Post Office doppelte Nachrichten auf. Der Standardwert lautet **`true`**.

IDCacheSize

Legt die Anzahl von Nachrichten-IDs fest, die das ID Cache halten sollte. Kommt es zu einem Server Failover, so wird auf das ID Cache als Teil des Prozesses zur Vermeidung des Versendens von doppelten Nachrichten nach einem Failover zugegriffen. Der Standardwert lautet **`500`** (Nachrichten).

PostOfficeName

Legt den Namen des Post Office fest.

NodeIDView

Dies gibt ein Set wieder, das die Node-IDs aller Knoten im Cluster enthält.

GroupName

Legt den Namen des Post Office fest, der mit anderen, identisch benannten Post Offices verbunden werden soll.

Anmerkung

Alle Post Offices im Cluster mit demselben Gruppennamen bilden gemeinsam ein Cluster. Vergewissern Sie sich, dass der Gruppenname bei sämtlichen Post Offices übereinstimmt, mit denen Sie ein Cluster bilden möchten.

Geclustert

Legt fest, ob das Post Office an einem Cluster teilnimmt, um distribuierte Warteschlangen und Topics zu bilden. Falls **`false`**, werden alle mit dem Cluster zusammenhängenden Attribute ignoriert.

StateTimeout

Legt die maximale Zeitdauer fest, die das Post Office wartet bis es den Gruppenstatus

empfängt, wenn ein Knoten zu einem bereits vorhandenen Cluster dazukommt. Der Standardwert lautet **30000** Millisekunden.

CastTimeout

Legt die maximale Zeitfest, die das Post Office synchron wartet bis es eine "Reply Casting Message" erhält. Der Standardwert lautet **30000** Millisekunden.

FailoverOnNodeLeave

Legt fest, wie Verbindungen gehandhabt werden, wenn ein Server-Knoten herunterfährt. Bei der Einstellung **true** findet ein Failover der Verbindungen des sauber terminierten Server-Knotens and einen anderen Knoten statt. Der Standardwert ist **false**.

MaxConcurrentReplications

Die maximale Anzahl nebenläufiger Replikationsanfragen, die getätigt werden soll, ehe wiederkehrende Antworten gesperrt werden. Dies soll verhindern, dass JGroups überschwemmt werden. Der Standardwert ist **50**.

Anmerkung

Sie sollten den Standardwert der MaxConcurrentReplications nicht ändern. Der Standardwert ist die optimale Einstellung.

ControlChannelConfig

Legt die JGroups Stack-Konfiguration für den Control-Channel fest. Der Control-Channel wird zum Senden von Anfragen/dem Erhalten von Antworten von anderen Knoten im Cluster verwendet

Anmerkung

JBoss Messaging verwendet JGroups für sämtliches Gruppen-Management. Es wird die standardmäßige JGroups-Konfiguration verwendet.

DataChannelConfig

Legt die JGroups Stack-Konfiguration für den Daten-Channel fest. Der Daten-Channel wird zum Senden von Anfragen/dem Erhalten von Nachrichten von anderen Nodes im Cluster und der Replikation von Session-Daten verwendet.

Anmerkung

JBoss Messaging verwendet JGroups für sämtliches Gruppen-Management. Es wird die standardmäßige JGroups-Konfiguration verwendet.

5.5. Konfiguration des Persistenz-Managers

JBoss Messaging wird mit einem JDBC Persistenz-Manager geliefert, der für den Umgang mit der Persistenz von Nachrichten-Daten in einer relationalen Datenbank verwendet wird, auf die via JDBC zugegriffen wird. Die Persistenz-Manager Implementierung ist einbindbar (pluggable) (der Persistenz-Manager ist ein Messaging Server Plug-In), wodurch es möglich wird, anderen Implementierungen die Persistierung von Nachrichten-Daten in nicht-relationalen Speichern, Dateispeichern usw. zu

ermöglichen.

Die Konfiguration "persistenter" Dienste wird in einer **<database type>-persistence-service.xml**-Datei gruppiert. Standardmäßig wird JBoss Messaging mit **hsqldb-persistence-service.xml** geliefert, die den Messaging Server zur Verwendung der standardmäßig in der JBoss Enterprise Application Server Instanz enthaltenen Hypersonic Datenbankinstanz konfiguriert.

Warnung

Hypersonic wird nicht für den Einsatz in Produktionsumgebungen unterstützt.

JBoss Messaging wird auch mit den bereits vorbereiteten Persistenz-Manager Konfigurationen für MySQL, Oracle, PostgreSQL, Sybase, MS SQL Server und DB2 geliefert. Die Beispielkonfigurationsdateien (wie **mysql-persistence-service.xml** und **ndb-persistence-service.xml**) sind im **jboss-as/docs/examples/jms**-Verzeichnis des Release-Bundles verfügbar.

Der JDBC Persistenz-Manager verwendet standardmäßiges SQL als seine Data Manipulation Language (DML), wird also eine Persistenz-Manager Konfiguration für einen anderen Datenbanktyp geschrieben, so muss die Data Definition Language (DDL) der Konfiguration geändert werden, die sich in der Regel auf pro-Datenbank-Basis unterscheidet.

JBoss Messaging bietet auch eine 'Null Persistence Manager' Konfiguration - diese kann verwendet werden, wenn Sie überhaupt keine Persistenz wünschen.

Der folgende Code ist die standardmäßige Hypersonic Persistenzmanager-Konfiguration:

```
<mbean code="org.jboss.messaging.core.jmx.JDBCPersistenceManagerService"
  name="jboss.messaging:service=PersistenceManager"
  xmbean-dd="xmdesc/JDBCPersistenceManager-xmbean.xml">

  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>

  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>

  <!-- The datasource to use for the persistence manager -->

  <attribute name="DataSource">java:/DefaultDS</attribute>

  <!-- If true will attempt to create tables and indexes on every start-up -->

  <attribute name="CreateTablesOnStartup">true</attribute>

  <!-- If true then will use JDBC batch updates -->

  <attribute name="UsingBatchUpdates">false</attribute>

  <!-- The maximum number of parameters to include in a prepared statement -->

  <attribute name="MaxParams">500</attribute>
</mbean>
```

Wichtig

Die maximale Größe von Sybase Datenbanktext und Bilddatentypen ist standardmäßig auf **2 kilobytes** gesetzt. Nachrichten, die diese Größe überschreiten, werden ohne weitere Informationen oder Warnung trunkiert. Setzen Sie den **@@TEXTSIZE**-Datenbankparameter auf einen höheren Wert, um eine mögliche Trunkierung zu vermeiden.
Trunkierung kann auch im Microsoft SQL Server vorkommen, wenn der **@@TEXTSIZE**-Wert auf einen geringeren als den Standardwert gesetzt ist. Weitere Informationen finden Sie unter http://jira.jboss.com/jira/browse/SOA-554.

Wichtig

Der Microsoft SQL Server gibt nicht automatisch Festplattenspeicher frei, wenn Daten aus einer Datenbank gelöscht werden. Wird der Festplattenspeicher als Datenspeicher für einen Dienst verwendet, der temporär viele Datensätze speichert (wie z.B. den Messaging Dienst), so wird der Festplattenplatz schnell viel größer als die tatsächlich gespeicherte Datenmenge. Datenbankadministratoren müssen Pläne zur Wartung der Datenbank implementieren, um sicherzustellen, dass unbenutzer Speicherplatz wieder freigegeben wird. In Ihrer Microsoft SQL Server Dokumentation finden Sie die DBCC-Befehle **ShrinkDatabase** und **UpdateUsage** zur Anleitung wie unbenutzter Speicherplatz wiedergewonnen wird. Weitere Informationen hierzu finden Sie unter https://jira.jboss.org/jira/browse/SOA-629

5.5.1. PersistenceManager MBean-Attribute

Die folgenden Attribute sind für die Konfiguration des Persistenz-Manager MBeans verfügbar:

CreateTablesOnStartup

Legt fest, ob Tabellen und Index bei Start des Persistenz Managers erstellt werden. Setzen Sie dies auf **true** (Standard), wenn Sie möchten, dass der Persistenz Manager beim Start die Tabellen (und Indizes) zu erstellen versucht. Falls die Tabellen (oder Indizes) bereits existieren, so wird eine **SQLException** durch den JDBC-Treiber gemeldet und vom Persistence Manager ignoriert, so dass fortgefahren werden kann.

UsingBatchUpdates

Legt fest, ob mehrere Datenbank-Updates in Batches gruppiert werden, um die Performance zu verbessern. Setzen Sie diesen Wert auf **true**, falls Ihre Datenbank JDBC-Batch-Updates unterstützt. Der Standardwert lautet **false**.

UsingBinaryStream

Legt fest, ob Nachrichten mit einem JDBC-Binärstrom gespeichert und gelesen werden, statt über **getBytes()** and **setBytes()**. Setzen Sie diesen Wert auf **false**, falls Ihre Datenbank **getBytes()** und **setBytes()** verwenden muss. Der Standardwert lautet **true**.

UsingTrailingByte

Legt fest, wie die Sybase Datenbank BLOBs mit Endnullen gehandhabt werden. Wenn auf **true** gesetzt, so wird ein nicht-Null Byte am Ende jedes BLOB vor Persistenz hinzugefügt und nach Persistenz aus dem BLOB entfernt, wodurch die Trunkierung der Datenbank verhindert wird. Der Standardwert lautet **false**

Anmerkung

Bestimmte Versionen von Sybase trunkieren ein BLOB mit Endnullen. Dieses Attribut wird nur benötigt, wenn Sie eine Sybase-Datenbank betreiben.

SupportsBlobOnSelect

Legt fest, wie BLOBs in bestimmte Datenbanktypen eingefügt werden. Ist dies auf **`false`** eingestellt, so wird eine 2-Etappen-Einfügung verwendet. Der Standardwert lautet **`true`**.

Anmerkung

Bestimmte Datenbanken, besonders Oracle, gestatten keine BLOB-Einfügung via einer **`INSERT INTO ... SELECT FROM`**-Anweisung und benötigen konditionale Nachrichteneinfügung in 2 Etappen. Setzen Sie dieses Attribute auf **`false`**, falls Sie eine Oracle Datenbank oder eine andere Datenbank mit dieser Anforderung betreiben.

SQLProperties

Hier sind DDL und DML für die bestimmte Datenbank festgelegt. Wird eine bestimmte DDL- oder DML-Anweisung nicht außer Kraft gesetzt, so wird die standardmäßige Hypersonic Konfiguration für diese Anweisung verwendet.

MaxParams

Legt die maximale Anzahl an Parametern fest, die pro vorbereiteter Anweisung gestattet sind, während Nachrichten geladen werden. Der Standardwert ist **`500`**.

UseNDBFailoverStrategy

Legt fest, ob eine SQL-Anweisung erneut ausgeführt wird, falls eine Datenbanktransaktionsfestschreibung in einer geclusterten Umgebung fehlschlägt. Falls auf **`true`** eingestellt, wird die SQL-Anweisung erneut ausgeführt falls die Festschreibung fehlschlägt. Erfolgt ein weiterer Fehler, so nimmt der Persistenzmanager an, dass dieser eine Folge der zuvor erfolgreich festgeschriebenen Transaktion ist und ignoriert diesen. Die Standardeinstellung dieses Attributs lautet **`false`**.

Anmerkung

Wenn einige Datenbanken, wie etwa MySQL, in geclusterten Umgebungen laufen, können sie während Datenbank-Transaktionsfestschreibungen fehlschlagen. Tritt dies ein, so ist der endgültige Transaktionsstatus unbekannt.

5.6. Konfiguration des JMS Nutzer-Managers

Der JMS Benutzer-Manager handhabt das Mapping vorkonfigurierter Client-IDs zu Benutzern und managt außerdem die Benutzer- und Rollentabellen, die - je nachdem welches Login-Modul Sie konfiguriert haben - verwendet werden oder auch nicht.

Hier ist eine Beispielkonfiguration für **`JMSUserManager`**:

```
<mbean code="org.jboss.jms.server.plugin.JDBCJMSUserManagerService"
  name="jboss.messaging:service=JMSUserManager"
  xmbean-dd="xmdesc/JMSUserManager-xmbean.xml">
  <depends>jboss.jca:service=DataSourceBinding,name=DefaultDS</depends>
  <depends optional-attribute-name="TransactionManager">
    jboss:service=TransactionManager
  </depends>
  <attribute name="DataSource">java:/DefaultDS</attribute>
  <attribute name="CreateTablesOnStartup">true</attribute>
  <attribute name="SqlProperties">
    CREATE_USER_TABLE=CREATE TABLE JBM_USER (USER_ID VARCHAR(32) NOT NULL,
      PASSWD VARCHAR(32) NOT NULL, CLIENTID VARCHAR(128),
      PRIMARY KEY(USER_ID)) ENGINE = INNODB

    CREATE_ROLE_TABLE=CREATE TABLE JBM_ROLE (ROLE_ID VARCHAR(32) NOT NULL,
      USER_ID VARCHAR(32) NOT NULL, PRIMARY KEY(USER_ID, ROLE_ID))
      ENGINE = INNODB

    SELECT_PRECONF_CLIENTID=SELECT CLIENTID FROM JBM_USER WHERE USER_ID=?

    POPULATE.TABLES.1=INSERT INTO JBM_USER (USER_ID,PASSWD,CLIENTID)
      VALUES ('jdoe','jdoepw','jdoe-id')
  </attribute>
</mbean>
```

5.6.1. JMSUserManager gemanagte Bean-Attribute

CreateTablesOnStartup

Legt fest, ob Tabellen und Index bei Start des JMSUserManager MBeans erstellt werden. Setzen Sie dies auf **`true`** (Standard), wenn Sie möchten, dass der JMSUserManager beim Start die Tabellen (und Indizes) zu erstellen versucht. Falls die Tabellen (oder Indizes) bereits existieren, so wird eine **`SQLException`** durch den JDBC-Treiber gemeldet und vom Persistenzmanager ignoriert, so dass fortgefahren werden kann.

UsingBatchUpdates

Legt fest, ob mehrere Datenbank-Updates in Batches gruppiert werden, um die Performance zu verbessern. Setzen Sie diesen Wert auf **`true`**, falls Ihre Datenbank JDBC-Batch-Updates unterstützt. Der Standardwert lautet **`false`**.

SQLProperties

Hier sind DDL und DML für die bestimmte Datenbank festgelegt. Wird eine bestimmte DDL- oder DML-Anweisung nicht außer Kraft gesetzt, so wird die standardmäßige Hypersonic Konfiguration für diese Anweisung verwendet.

Sie können auch standardmäßige Property-Daten für Nutzer und Rollen einfügen, indem Sie den Daten **`POPULATE.TABLES`** voranstellen.

5.7. Konfiguration von Destinationen

5.7.1. Vorkonfigurierte Destinationen

JBoss Messaging wird mit einem Standardsatz vorkonfigurierter Destinationen geliefert, die während des Server Start-ups deployt werden. Die Datei, die die Konfiguration für diese Destinationen enthält, ist im folgenden Abschnitt von **`destinations-service.xml`** angeführt:

```
<!--
      The Default Dead Letter Queue. This destination is a dependency of an EJB MDB
container.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=DLQ"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-
name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>

   <!--
      The Default Expiry Queue.
   -->

   <mbean code="org.jboss.jms.server.destination.QueueService"
      name="jboss.messaging.destination:service=Queue,name=ExpiryQueue"
      xmbean-dd="xmdesc/Queue-xmbean.xml">
      <depends optional-attribute-
name="ServerPeer">jboss.messaging:service=ServerPeer</depends>
      <depends>jboss.messaging:service=PostOffice</depends>
   </mbean>
```

5.7.2. Konfiguration von Warteschlangen

5.7.2.1. Warteschlangen MBean-Attribute

Name

Definiert den Namen der Warteschlange.

JNDIName

Definiert den JNDI-Namen, der die Warteschlange bindet.

DLQ

Die für diese Warteschlange verwendete DLQ (Dead Letter Queue). Setzt einen in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

ExpiryQueue

Definiert die für diese Warteschlange verwendete Expiry-Queue. Setzt einen in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

RedeliveryDelay

Definiert die Verzögerung der erneuten Auslieferung ("Redelivery Delay"), die für diese Warteschlange verwendet werden soll. Setzt den in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

MaxDeliveryAttempts

Die maximale Anzahl von Malen, die der Versuch der Auslieferung einer Nachricht unternommen wird ehe die Nachricht an die DLQ geschickt wird (falls konfiguriert). Falls auf **`-1`** (der Standard) gesetzt, wird der Wert der ServerPeer Konfigurationsdatei verwendet. Jede andere Einstellung setzt den in der ServerPeer Konfigurationsdatei gesetzten Wert außer Kraft.

CreatedProgrammatically

Gibt **`true`** wieder, wenn die Warteschlange programmatisch erstellt wurde.

MessageCount

Gibt die Gesamtzahl von Nachrichten in der Warteschlange wieder. Das heißt die Anzahl der terminierten plus die Anzahl der ausgelieferten plus die Anzahl der nicht ausgelieferten.

ScheduledMessageCount

Gibt die Anzahl *terminierter Nachrichten* in der Warteschlange wieder. Dies ist die Anzahl von Nachrichten, die zur Auslieferung zu einem späteren Zeitpunkt terminiert sind.

Terminierte Auslieferung ist ein Feature von JBoss Messaging, bei dem Sie eine Nachricht senden und festlegen können, wann der früheste Zeitpunkt sein soll, an dem diese ausgeliefert wird. Zum Beispiel können Sie eine Nachricht jetzt senden, diese wird jedoch erst in 2 Stunden ausgeliefert. Um dies zu tun, stellen Sie folgendes Im Message-Header ein:

```
long now = System.currentTimeMillis();

Message msg = sess.createMessage();

msg.setLongProperty(JBossMessage.JMS_JBOSS_SCHEDULED_DELIVERY_PROP_NAME,
  now + 1000 * 60 * 60 * 2);

prod.send(msg);
```

MaxSize

Eine maximale Größe (in Form der Nachrichtenzahl) kann für eine Warteschlange festgelegt werden. Alle danach ankommenden Nachrichten werden fallengelassen. Der Standard lautet **-1**, welches ungebunden ist.

Geclustert

Dieses Attribute muss auf **true** gesetzt sein, wenn das Ziel geclustert ist.

MessageCounter

Jede Warteschlange wartet einen Nachrichtenzähler.

MessageCounterStatistics

Die Statistiken für den Nachrichtenzähler.

MessageCounterHistoryDayLimit

Die maximale Anzahl von Tagen, die die Nachrichtenzähler-Historie gehalten wird. Setzt jeden in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

ConsumerCount

Die Anzahl von Verbrauchern, die zum aktuellen Zeitpunkt von der Warteschlange zehrt.

DropOldMessageOnRedeploy

Legt fest wie Warteschlangendienste mit geclusterten Attributen, die sich von zuvor deployten Attributen unterscheiden, gehandhabt werden. Bei der Einstellung **true** werden alle übrig gebliebenen Nachrichten in der Warteschlange nach dem erneuten Deployment des Warteschlangendienstes gelöscht, wenn das Warteschlangendienstattribut ein anders geclustertes Attribut enthält. Bei Einstellung auf **false** (Standard), sind alle Nachrichten vorbehalten.

Warnung - Erwägungen bei erneutem Deployment

Wenn Sie eine Destination erneut deployen, müssen Sie alle Nodes im Cluster herunterfahren, ordnungsgemäße Konfigurationsänderungen durchführen und die Nodes dann erneut starten.
Das Redeployment von einer nicht-geclusterten zu einer geclusterten Warteschlange erfordert es, dass Sie das geclusterte Attribut auf **`true`** setzen und die Konfiguration des Warteschlangendienstes jedem der Nodes hinzufügen.
Das Redeployment von einer geclusterten zu einer nicht-geclusterten Warteschlange erfordert es, dass Sie das geclusterte Attribut in einer der Warteschlangenkonfigurationen auf **`false`** setzen und alle anderen Warteschlangen im Cluster löschen.

5.7.2.1.1. Destination-Sicherheitskonfiguration

<SecurityConfig> legt fest, welche Rollen an der Destination lesen, schreiben und erstellen dürfen. Sie verwendet dieselbe Syntax und Semantik wie die Sicherheitskonfiguration in JBossMQ Destinationen.

```
<SecurityConfig>
  <security>
    <role read="true" write="true" create="true"/>
  </security>
<SecurityConfig>
```

Das <SecurityConfig>-Element muss ein <security>-Element enthalten, welches mehrere <role>-Elemente enthalten kann. Ein <role>-Element definiert den Zugriff für diese bestimmte Rolle unter Verwendung folgender Attribute:

`read`

Legt fest, dass die Rolle Verbraucher erstellen, Nachrichten empfangen und die Destination browsen kann.

`write`

Legt fest, dass die Rolle Producer erstellen oder Nachrichten an die Destination schicken kann.

`create`

Legt fest, dass die Rolle beständige Abonnements an dieser Destination erstellen kann.

Anmerkung

Die Sicherheitskonfiguration für eine Destination ist optional. Ist kein **`SecurityConfig`**-Element festgelegt, so wird die standardmäßige Sicherheitskonfiguration des Server Peer verwendet.

5.7.2.1.2. Destination Paging Parameter

Pageable Channels ist ein JBoss Messaging Feature, dass es Ihen gestattet, die maximale Anzahl von gleichzeitig im Speicher gespeicherten Nachrichten auf einer Warteschlange-per-Warteschlange oder Topic-per-Topic Basis festzulegen. JBoss Messaging pagt dann transparent in Blöcken Nachrichten zum und vom Speicher, und gestattet Warteschlangen auf sehr große Größen anzuwachsen, ohne dass es mit Anwachsen der Channel-Größe zu einer Performance-Degradierung käme.

Die einzelnen, mit pagebaren Channels assoziierten Parameter lauten wie folgt:

`FullSize`

Legt die maximale Anzahl von Nachrichten, die gleichzeitig in der Warteschlange oder Topic-Abonnements gehalten wird. Die tatsächliche Warteschlange kann mehr Nachrichten als dies enthalten, aber diese sind von und zum Speicher paginiert, wie nötig, da Nachrichten

hinzugefügt oder verbraucht werden. Ist kein Wert festgelegt, so lautet der Standard **75000**.

PageSize

Legt die maximale Anzahl von Nachrichten fest, die pro Operation vor-geladen werden, wenn Nachrichten von der Warteschlange oder dem Abonnement geladen werden. Ist kein Wert festgelegt, so lautet der Standard **2000**.

DownCacheSize

Legt die maximale Anzahl von Nachrichten fest, die das Down Cache enthält, ehe die Nachrichten an den Speicher gesendet werden. Der Standardwert lautet **2000** Nachrichten.

Beim Paginieren von Nachrichten von der Warteschlange zum Speicher landen diese zunächst in einem *Down Cache* ehe sie in den Speicher geschrieben werden. Dies ermöglicht es, dass das Schreiben als einzelne Operation durchzuführen, was wiederum die Performance verbessert.

Anmerkung

Falls Sie die für temporäre Warteschlangen verwendeten Paging Parameter festlegen wollen, so müssen Sie diese an der entsprechenden Connection Factory festlegen. Einzelheiten zu den verschiedenen Connection Factories finden Sie unter Abschnitt 5.8, „Konfiguration von "Connection Factories"“.

5.7.2.1.3. Warteschlangen-gemanagte Bean Operationen

RemoveAllMessages

Entfernt (und löscht) alle Nachrichten aus der Warteschlange.

Wichtig

Dies löscht dauerhaft alle Nachrichten aus der Warteschlange. Verwenden Sie diese Operation mit Vorsicht.

ListAllMessages

Listet alle zum aktuellen Zeitpunkt in der Warteschlange befindlichen Nachrichten. Die Verwendung eines JMS-Selektors als Argument in dieser Operation gestattet den Abruf eines Untersatzes an Nachrichten in der Warteschlange, die mit den gegebenen Kriterien übereinstimmen.

ListDurableMessages

Listet alle *beständigen* Nachrichten in der Warteschlange. Durch Verwendung des JMS-Selektors können Se einen Untersatz der Nachrichten in der Warteschlange abrufen, auf die die Kriterien zutreffen.

ListNonDurableMessages

Listet alle *nicht beständigen* Nachrichten in der Warteschlange. Durch Verwendung des JMS-Selektors als Argument können Sie einen Untersatz der Nachrichten in der Warteschlange abrufen, auf die die Kriterien zutreffen.

ResetMessageCounter

Setzt den Nachrichtenzähler auf Null zurück.

ResetMessageCounterHistory

Setzt die Historie des Nachrichtenzählers zurück.

ListMessageCounterAsHTML

Listet den Nachrichtenzähler in HTML-Format

ListMessageCounterHistoryAsHTML

Listet die Historie des Nachrichtenzählers in HTML-Format

5.7.3. Konfiguration von Topics

5.7.3.1. Topic gemanagte Bean-Attribute

Name

Definiert den Namen des Topic

JNDIName

Definiert den JNDI-Namen, wo das Topic gebunden ist

DLQ

Definiert die für dieses Topic verwendete Dead Letter Queue (DLQ). Setzt jeden in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

ExpiryQueue

Definiert die für dieses Topic verwendete Expiry-Warteschlange. Setzt jeden in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

RedeliveryDelay

Definiert die für dieses Topic verwendete Verzögerung für die erneute Auslieferung ("Redelivery Delay"). Setzt jeden in der ServerPeer-Konfigurationsdatei eingestellten Wert außer Kraft.

MaxDeliveryAttempts

Die maximale Anzahl von Malen, die der Versuch der Auslieferung einer Nachricht unternommen wird ehe die Nachricht an die DLQ geschickt wird (falls konfiguriert). Falls auf **`-1`** (der Standard) gesetzt, wird der Wert der ServerPeer Config verwendet. Jede andere Einstellung setzt den in der ServerPeer Config gesetzten Wert außer Kraft.

CreatedProgrammatically

Gibt **`true`** wieder, wenn das Topic programmatisch erstellt wurde.

MaxSize

Legt eine maximale Größe (in Form der Nachrichtenzahl) für ein Topic-Abonnement fest. Alle danach ankommenden Nachrichten werden fallengelassen. Der Standard lautet **`-1`**, welches keine Größenbeschränkung bedeutet.

Geclustert

Setzen Sie dies auf **`true`**, falls Ihr Ziel geclustert ist.

MessageCounterHistoryDayLimit

Definiert die maximale Anzahl von Tagen, die die Nachrichtenzähler-Historie gehalten wird. Setzt jeden im ServerPeer eingestellten Wert außer Kraft.

MessageCounters

Wiedergabe einer Liste von Nachrichtenzählern für die Abonnements dieses Topics.

AllMessageCount

Wiedergabe der Gesamtzahl an Nachrichten in allen Abonnements dieses Topics.

DurableMessageCount

Wiedergabe der Gesamtzahl von *dauerhaften* Nachrichten in allen Abonnements dieses Topics.

NonDurableMessageCount

Wiedergabe der Gesamtzahl von *nicht-dauerhaften* Nachrichten in allen Abonnements dieses Topics.

DropOldMessageOnRedeploy

Legt fest wie Warteschlangendienste mit geclusterten Attributen, die sich von zuvor deployten Attributen unterscheiden, gehandhabt werden. Bei der Einstellung **`true`** werden alle übrig gebliebenen Nachrichten in der Warteschlange nach dem erneuten Deployment des Warteschlangendienstes gelöscht, wenn das Warteschlangendienstattribut ein anders geclustertes Attribut enthält. Bei Einstellung auf **`false`** (Standard), sind alle Nachrichten vorbehalten.

Warnung - Erwägungen bei erneutem Deployment

Wenn Sie eine Destination erneut deployen, müssen Sie alle Nodes im Cluster herunterfahren, ordnungsgemäße Konfigurationsänderungen durchführen und die Nodes dann erneut starten.
Das Redeployment von einer nicht-geclusterten zu einer geclusterten Warteschlange erfordert es, dass Sie das geclusterte Attribut auf **`true`** setzen und die Konfiguration des Warteschlangendienstes jedem der Nodes hinzufügen.
Das Redeployment von einer geclusterten zu einer nicht-geclusterten Warteschlange erfordert es, dass Sie das geclusterte Attribut in einer der Warteschlangenkonfigurationen auf **`false`** setzen und alle anderen Warteschlangen im Cluster löschen.

AllSubscriptionsCount

Wiedergabe der Gesamtzahl aller Abonnements dieses Topics.

DurableSubscriptionsCount

Wiedergabe der Gesamtzahl aller dauerhaften Abonnements dieses Topics.

NonDurableSubscriptionsCount

Wiedergabe der Gesamtzahl aller nicht dauerhaften Abonnements dieses Topics.

5.7.3.1.1. Destination-Sicherheitskonfiguration

<SecurityConfig> legt fest, welche Rollen an der Destination lesen, schreiben und erstellen dürfen. Sie verwendet dieselbe Syntax und Semantik wie die Sicherheitskonfiguration in JBossMQ Destinationen.

Das <SecurityConfig>-Element muss ein <security>-Element enthalten, welches mehrere <role>-Elemente enthalten kann. Ein <role>-Element definiert den Zugriff für diese bestimmte Rolle unter Verwendung folgender Attribute:

read
: Legt fest, dass die Rolle Verbraucher erstellen, Nachrichten empfangen und die Destination browsen kann.

write
: Legt fest, dass die Rolle Producer erstellen oder Nachrichten an die Destination schicken kann.

create
: Legt fest, dass die Rolle beständige Abonnements an dieser Destination erstellen kann.

Anmerkung

Die Sicherheitskonfiguration für eine Destination ist optional. Ist kein **`SecurityConfig`**-Element festgelegt, so wird die standardmäßige Sicherheitskonfiguration des Server Peer verwendet.

5.7.3.1.2. Destination Paging Parameter

In der Vergangenheit musste eine Warteschlange komplett im Speicher gespeichert sein, damit eine Applikation eine Warteschlange oder ein Abonnement unterstützen konnte. Dies war bei sehr langen Warteschlangen oder Abonnements nicht immer möglich.

Pageable Channels ist ein neues JBoss Messaging Feature mit dem Sie die maximale Anzahl von gleichzeitig im Speicher gespeicherten Nachrichten auf einer Warteschlange-per-Warteschlange oder Topic-per-Topic Basis festlegen können. JBoss Messaging pagt dann transparent in Blöcken Nachrichten zum und vom Speicher, und gestattet Warteschlangen und Abonnements auf sehr große Größen anzuwachsen, ohne dass es mit Anwachsen der Channel-Größe zu einer Performance-Degradierung käme. Dies wurde mit Warteschlangen mit über 10 Millionen 2 Kilobyte Nachrichten auf sehr einfacher Hardware getestet, und hat das Potenzial entsprechend größere Nachrichtenzahlen zu erreichen.

Die einzelnen, mit pagebaren Channels assoziierten Parameter lauten wie folgt:

FullSize
: Legt die maximale Anzahl von Nachrichten, die gleichzeitig in der Warteschlange oder Topic-Abonnements gehalten wird. Die tatsächliche Warteschlange kann mehr Nachrichten als dies enthalten, aber diese sind von und zum Speicher paginiert, wie nötig, da Nachrichten hinzugefügt oder verbraucht werden. Ist kein Wert festgelegt, so lautet der Standard **`75000`**.

PageSize
: Legt die maximale Anzahl von Nachrichten fest, die pro Operation vor-geladen werden, wenn Nachrichten von der Warteschlange oder dem Abonnement geladen werden. Ist kein Wert festgelegt, so lautet der Standard **`2000`**.

DownCacheSize
: Legt die maximale Anzahl von Nachrichten fest, die das Down Cache enthält, ehe die Nachrichten an den Speicher gesendet werden. Der Standardwert lautet **`2000`** Nachrichten.

  Beim Paginieren von Nachrichten von der Warteschlange zum Speicher landen diese zunächst in einem *Down Cache* ehe sie in den Speicher geschrieben werden. Dies ermöglicht es, dass das Schreiben als einzelne Operation durchzuführen, was wiederum die Performance verbessert.

Anmerkung

Paging Parameter für temporäre Warteschlangen müssen an der entsprechenden Connection Factory festgelegt werden. Weitere Informationen finden Sie im Abschnitt zur Connection Factory Konfiguration.

5.7.3.2. Topic gemanagte Bean Operationen

RemoveAllMessages

Alle Nachrichten von Abonnements dieses Topics entfernen (und löschen).

Wichtig

Mit Vorsicht verwenden. Es löscht dauerhaft alle Nachrichten aus dem Topic.

ListAllMessages

Listet alle zu einem bestimmten Abonnement gehörenden Nachrichten. Durch Verwendung des JMS-Selektors können Sie einen Untersatz der Nachrichten in der Warteschlange abrufen, auf die die Kriterien zutreffen.

ListDurableMessages

Listet alle beständigen Nachrichten, die zu einem bestimmten Abonnement gehören. Durch Verwendung des JMS- Selektors können Sie einen Untersatz der Nachrichten in der Warteschlange abrufen, auf die die Kriterien zutreffen.

ResetMessageCounter

Setzt den Nachrichtenzähler auf Null zurück.

ResetMessageCounterHistory

Setzt die Historie des Nachrichtenzählers zurück.

ListAllSubscriptionsAsHTML

Listet alle Abonnements für dieses Topic in HTML-Format.

ListDurableSubscriptionsAsHTML

Listet alle dauerhaften Abonnements für dieses Topic in HTML-Format.

ListNonDurableSubscriptions

Listet alle nicht-beständigen Nachrichten, die zu einem bestimmten Abonnement gehören. Durch Verwendung des JMS-Selektors können Sie einen Untersatz der Nachrichten in der Warteschlange abrufen, auf die die Kriterien zutreffen.

ListNonDurableSubscriptionsAsHTML

Listet alle nicht dauerhaften Abonnements für dieses Topic in HTML-Format.

5.8. Konfiguration von "Connection Factories"

Bei der Standardkonfiguration bindet JBoss Messaging beim Start-up zwei Connection Factories in JNDI.

Die erste Connection Factory ist die standardmäßige nicht-geclusterte Connection Factory. Diese Connection Factory soll die Kompatibilität mit Anwendungen wahren, die ursprünglich für JBoss MQ geschrieben wurden, das kein automatisches Failover oder Lastverteilung bietet. Diese Connection Factory sollte verwendet werden, wenn Sie auf Client-Seite kein automatisches Failover oder Lastverteilung benötigen.

Die erste Connection Factory wird in die folgenden JNDI-Kontexte gebunden:

- **`/ConnectionFactory`**
- **`/XAConnectionFactory`**
- **`java:/ConnectionFactory`**
- **`java:/XAConnectionFactory`**.

Die zweite Connection Factory ist die standardmäßige, geclusterte Connection Factory, die in die folgenden JNDI-Kontexte gebunden wird:

- **`/ClusteredConnectionFactory`**
- **`/ClusteredXAConnectionFactory`**
- **`java:/ClusteredConnectionFactory`**
- **`java:/ClusteredXAConnectionFactory`**

Falls Sie eine standardmäßige Client ID für eine Connection Factory bieten wollen oder Sie diese an unterschiedlichen Stellen in JNDI binden wollen, so konfigurieren und deployen Sie zusätzliche Connection Factories. Das Deployment einer neuen Connection Factory ist äquivalent mit dem Hinzufügen einer neuen **`ConnectionFactory`**-MBean-Konfiguration zu **`connection-factories-service.xml`**.

Es ist auch möglich einen ganz neuen Service-Deployment-Deskriptor ***`<name>`*`-service.xml`** zu erstellen und diesen in **`$JBOSS_HOME/server/messaging/deploy`** zu deployen.

Aktivierung des Supports für automatischen Failover oder Lastverteilung durch Einstellung der entsprechenden Attribute in Ihrer Connection Factory:

Beispiel 5.1. Connection Factory

Dieses Connection Factory Beispiel erstellt eine Connection Factory mit der vorkonfigurierten Client ID **`myClientID`**, die an zwei Orte im JNDI-Baum gebunden ist: **`/MyConnectionFactory`** und **`/factories/cf`**.

Das Beispiel setzt die folgenden Standardwerte außer Kraft:

- **`PreFetchSize`**
- **`DefaultTempQueueFullSize`**
- **`DefaultTempQueuePageSize`**
- **`DefaultTempQueueDownCacheSize`**
- **`DupsOKBatchSize`**
- **`SupportsFailover`**
- **`SupportsLoadBalancing`**
- **`LoadBalancingFactory`**

Die Connection Factory verwendet den standardmäßigen Remoting Connector. Um einen anderen Remoting Connector mit der Connection Factory zu verwenden, ändern Sie das **`Connector`**-Attribut um den Dienstnamen des Connectors festzulegen, den Sie verwenden möchten.

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory"
  name="jboss.messaging.connectionfactory:service=MyConnectionFactory"
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>jboss.messaging:service=PostOffice</depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/factories/cf</binding>
    </bindings>
  </attribute>

  <attribute name="ClientID">myClientID</attribute>

  <attribute name="SupportsFailover">true</attribute>

  <attribute name="SupportsLoadBalancing">false</attribute>

  <attribute name="LoadBalancingFactory">
    org.acme.MyLoadBalancingFactory
  </attribute>

  <attribute name="PrefetchSize">1000</attribute>

  <attribute name="SlowConsumers">false</attribute>

  <attribute name="StrictTck">true</attribute>

  <attribute name="SendAcksAsync">false</attribute>

  <attribute name="DefaultTempQueueFullSize">50000</attribute>

  <attribute name="DefaultTempQueuePageSize">1000</attribute>

  <attribute name="DefaultTempQueueDownCacheSize">1000</attribute>

  <attribute name="DupsOKBatchSize">10000</attribute>
</mbean>
```

5.8.1. ConnectionFactory gemanagte Bean Attribute

ClientID

Connection Factories können mit einer Client-ID vorkonfiguriert werden. Alle Verbindungen, die unter Verwendung dieser Connection Factory erstellt wurden, erhalten diese Client-ID

JNDIBindings

De Liste der verfügbaren JNDI-Bindings für diese Connection Factory

PrefetchSize

Dieser Parameter legt die Fenstergröße für die Verbraucherflusskontrolle fest. Die Fenstergröße bestimmt die Anzahl von Nachrichten, die ein Server an einen Verbraucher ("Consumer") ohne Blockierung senden kann. Jeder Consumer wartet einen Zwischenspeicher (Buffer) an Nachrichten, von denen er sich speist.

TCP implementiert seine eigene Flusskontrolle , d.h. wenn Sie es auf eine zu hohe Zahl setzen, so kann die TCP-Fenstergröße vor ***PrefetchSize*** erreicht werden, was beim Schreiben zu

Blockierungen führen kann.

SlowConsumers

Legt fest, ob die gestattbare Zwischenspeichergröße für langsame Verbraucher gesenkt wird. Die Reduktion der Zwischenspeichergröße für langsame Verbraucher führt dazu, dass Nachrichten von schnelleren Verbrauchern konsumiert werden können. Es ist nicht möglich, den Zwischenspeicher komplett zu deaktivieren, jedoch kann die Einstellung des **`SlowConsumers`**-Attributs auf **`true`** die Zwischenspeichergröße reduzieren. Die Einstellung dieses Attributs auf **`true`** ist äquivalent zur Einstellung **`PrefetchSize`** to **`1`**, welches der niedrigste mögliche Wert ist.

StrictTck

Aktiviert striktes JMS-Verhalten, wenn das Attribut auf **`true`** gesetzt ist. Striktes JMS-Verhalten wird vom Technology Compatibility Kit (TCK) erfordert.

SendAcksAsync

Legt fest, dass Bestätigungen asynchron gesendet werden, wenn das Attribut auf **`true`** gesetzt ist. Dies kann die Performance verbessern, insbesondere dann, wenn der **`auto_acknowledge`**-Modus aktiv ist.

DefaultTempQueueFullSize

Optionales Attribut, das die Paging Parameter für temporäre Warteschlangendestinationen voller Größe festlegt, die auf mit dieser Connection Factory erstellte Verbindungen beschränkt sind. Der Standardwert ist **`200000`**. Weitere Informationen zu diesen Attributen finden Sie unter Abschnitt 5.7.3.1.2, „Destination Paging Parameter“.

DefaultTempQueuePageSize

Optionales Attribut, das die Paging Parameter für temporäre Seitengrößendestinationen festlegt, die auf mit dieser Connection Factory erstellte Verbindungen beschränkt sind. Der Standardwert ist **`2000`**. Weitere Informationen zu diesen Attributen finden Sie unter Abschnitt 5.7.3.1.2, „Destination Paging Parameter“.

DefaultTempQueueDownCacheSize

Optionales Attribut, das die Paging Parameter für temporäre Down Cache Größen Destinationen festlegt, die auf mit dieser Connection Factory erstellte Verbindungen beschränkt sind. Der Standardwert ist **`2000`**. Weitere Informationen zu diesen Attributen finden Sie unter Abschnitt 5.7.3.1.2, „Destination Paging Parameter“.

DupsOKBatchSize

Legt die Anzahl von **`DUPS_OK_ACKNOWLEDGE`** Bestätigungen fest, die lokal gepuffert werden ehe sie versendet werden. Der Standardwert lautet **`2000`**.

SupportsLoadBalancing

Legt fest, ob die Lastverteilung auf Client-Seite an einer Connection Factory an geclusterten Installationen aktiviert ist. Ist die Lastverteilung aktiviert so wird jede von dieser Connection Factory erstellte Verbindung über die Nodes eines Clusters hinweg lastverteilt. Wird eine Verbindung an einem bestimmten Node erstellt, so bleibt sie an diesem Node. Der Standardwert lautet **`false`**.

SupportsFailover

Ist an einer Connection Factory automatischer Failover aktiviert, so erfolgt durch JBoss Messaging ein automatischer und transparenter Failover an einen anderen Node des Clusters, falls ein Verbindungsproblem auftritt. Der Standardwert ist **`false`**.

Anmerkung

Ist der automatische Failover deaktiviert, so obliegt es der Verantwortung des Benutzers Verbindungsausnahmen in synchronen JMS-Operations aufzufangen und einen JMS **`ExceptionListener`** zu installieren und Ausnahmen asynchron abzufangen. Wird eine Ausnahme aufgefangen, so sollte der Code auf Client-Seite mittels HAJNDI eine neue Connection-Factory suchen und mittels derer die Verbindung wiederherstellen.

DisableRemotingChecks

Legt fest, ob die Connection Factory überprüft ob der entsprechende JBoss Remoting Connector "vernünftige" Werte besitzt. JBoss Messaging ist hinsichtlich seiner Werte sehr empfindlich und bei vielen von diesen besteht eigentlich selten Grund sie zu verändern. Um diese Überprüfung auf "Vernunft" der Werte abzustellen, setzen Sie **`DisableRemotingChecks`** auf **`false`**. Der Standardwert ist **`true`**.

Warnung

Deaktivieren Sie die Remoting Überprüfungen nicht; Systeminstabilität.

LoadBalancingFactory

Bestimmt die Lastverteilungs-Factory-Implementierung auf Client-Seite, die die Connection Factory verwendet. Der Wert muss dem Namen einer Klasse entsprechen, die das Interface **`org.jboss.jms.client.plugin.LoadBalancingFactory`** implementiert.

Der Standardwert ist **`org.jboss.jms.client.plugin.RoundRobinLoadBalancingFactory`**, welcher nach Round-Robin Art Verbindungen über ein Cluster lastverteilt

Connector

Legt den von der Connection Factory verwendeten Remoting Connector fest. Verschiedene Connection Factories können verschiedene Connectors verwenden - Sie könnten etwa eine Connection Factory deployen, die HTTP-Transport zur Kommunikation mit dem Server verwendende Verbindungen erstellt und eine weitere, die "Bisocket"-Transport zur Kommunikation verwendende Verbindungen erstellt.

EnableOrderingGroup

Legt fest, ob strenge Nachrichtenreihenfolge ("strict message ordering") an der **`ConnectionFactory`** aktiviert ist. Wenn auf **`true`** eingestellt, werden Nachrichten die von durch die aktivierte Connection Factory erstellten Producern verschickt werden "Ordering Group" Nachrichten. Der Standardwert für diesen Parameter ist **`false`**.

DefaultOrderingGroupName

Legt den Standardnamen der "Message Ordering Group" fest. Der festgelegte Name wird wirksam, sobald der **`EnableOrderingGroup`**-Parameter auf **`true`** gesetzt ist. Fehlt dieses Attribut, so wird der Gruppenname automatisch generiert.

5.9. Konfiguration des Remoting Connectors

JBoss Messaging verwendet JBoss Remoting für sämtliche Kommunikation zwischen dem Client und dem Server.

Weitere Informationen zur Konfiguration und den Möglichkeiten von JBoss Remoting finden Sie im [Remoting]-Kapitel im *JBoss Administration and Configuration Guide*.

Die Standardkonfiguration beinhaltet einen einzelnen Remoting Connector, der von der einzelnen Standard Connection Factory verwendet wird. Jede Connection Factory kann so konfiguriert werden, dass Sie ihren eigenen Connector benutzt.

Der Standard-Connector wird für die Verwendung des Remoting Bisocket-Transports konfiguriert. Bei Bisocket-Transport handelt es sich um einen TCP Socket-basierten Transport, der nur auf Server-Seite auf Verbindungen horcht und diese annimt. D.h. Verbindungen werden immer durch die Client-Seite initiiert. Dies bedeutet, dass es gut in typischen Firewall-Szenarien funktioniert, bei denen nur eingehende Verbindungen auf dem Server gestattet sind oder aber bei denen nur ausgehende Verbindungen vom Client gestattet sind.

Der Bisocket-Transport kann so konfiguriert werden, dass SSL verwendet wird, wenn eine höhere Sicherheitsebene benötigt wird.

Beim anderen unterstützten Transport handelt es sich um HTTP-Transport. Dieser verwendet das HTTP-Protokoll zur Kommunikation zwischen Client und Server. Daten werden am Client empfangen, indem der Client periodisch den Server auf Nachrichten pollt. Dieser Transport eignet sich gut für Situationen, in denen eine Firewall zwischen Client und Server existiert, die nur eingehenden HTTP-Traffic auf dem Server gestattet. Bitte beachten Sie, dass diese Art von Transport aufgrund des Wesens des Pollings und des HTTP-Protokolls nicht so performant wie der Bisocket-Transport ist. Bitte beachten Sie außerdem, dass sie auch nicht für Situationen mit großer Lasthöhe eignen.

Derzeit werden keine anderen Remoting Transports von JBoss Messaging unterstützt

Konfigurationsinformationen zum Remoting finden Sie in ***$JBOSS_HOME*/server/*$SERVER*/deploy/messaging/remoting-bisocket-service.xml**. Der folgende Code ist ein Beispiel für eine Bisocket Remoting Konfiguration:

Beispiel 5.2. Bisocket Remoting-Konfiguration:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
     Standard bisocket-based Remoting service deployment descriptor.

     $Id: remoting-bisocket-service.xml 3981 2008-03-28 18:00:41Z timfox $
 -->

<server>

   <!-- Standard bisocket connector - the bisocket transport only opens
connection from client->server
        so can be used with firewalls where only outgoing connections are
allowed.
        For examples of HTTP and SSL transports see docs/examples -->
   <mbean code="org.jboss.remoting.transport.Connector"
          name="jboss.messaging:service=Connector,transport=bisocket"
          display-name="Bisocket Transport Connector">
      <attribute name="Configuration">
         <config>
            <invoker transport="bisocket">

               <!-- There should be no reason to change these parameters -
warning!
                    Changing them may stop JBoss Messaging working correctly --
>
               <attribute name="marshaller"
isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="unmarshaller"
isParam="true">org.jboss.jms.wireformat.JMSWireFormat</attribute>
               <attribute name="dataType" isParam="true">jms</attribute>
               <attribute name="socket.check_connection"
isParam="true">false</attribute>
               <attribute
name="serverBindAddress">${jboss.bind.address}</attribute>
               <attribute
name="serverBindPort">${jboss.messaging.connector.bisocket.port:4457}</attribute>
               <attribute name="clientSocketClass"
isParam="true">org.jboss.jms.client.remoting.ClientSocketWrapper</attribute>
               <attribute
name="serverSocketClass">org.jboss.jms.server.remoting.ServerSocketWrapper</attr
ibute>
               <attribute
name="onewayThreadPool">org.jboss.jms.server.remoting.DirectThreadPool</attribut
e>

               <!-- the following parameters are useful when there is a
firewall between client and server. Uncomment them if so.-->
               <!--
               <attribute name="numberOfCallRetries"
isParam="true">1</attribute>
               <attribute name="pingFrequency"
isParam="true">214748364</attribute>
               <attribute name="pingWindowFactor" isParam="true">10</attribute>
               <attribute name="generalizeSocketException"
isParam="true">true</attribute>
               -->

               <!-- Now remoting supports socket write timeout configuration.
Uncomment this if you need it. -->
               <!--
               <attribute name="writeTimeout" isParam="true">30000</attribute>
               -->

               <!-- End immutable parameters -->

               <attribute name="stopLeaseOnFailure"
isParam="true">true</attribute>
```

```
               <!-- Periodicity of client pings. Server window by default is
twice this figure -->
               <attribute name="clientLeasePeriod"
isParam="true">10000</attribute>
               <attribute name="validatorPingPeriod"
isParam="true">10000</attribute>
               <attribute name="validatorPingTimeout"
isParam="true">5000</attribute>

               <attribute name="failureDisconnectTimeout"
isParam="true">0</attribute>
               <attribute name="callbackErrorsAllowed">1</attribute>
               <attribute name="registerCallbackListener">false</attribute>
               <attribute name="useClientConnectionIdentity"
isParam="true">true</attribute>

             <attribute name="timeout" isParam="true">0</attribute>

               <!-- Max Number of connections in client pool. This should be
significantly higher than
                    the max number of sessions/consumers you expect -->
               <attribute name="JBM_clientMaxPoolSize"
isParam="true">200</attribute>

               <!-- The maximum time to wait before timing out on trying to
write a message to socket for delivery -->
               <attribute name="callbackTimeout">10000</attribute>

               <!-- Use these parameters to specify values for binding and
connecting control connections to
                    work with your firewall/NAT configuration
               <attribute name="secondaryBindPort">xyz</attribute>
               <attribute name="secondaryConnectPort">abc</attribute>
               -->

            </invoker>
            <handlers>
               <handler
subsystem="JMS">org.jboss.jms.server.remoting.JMSServerInvocationHandler</handle
r>
            </handlers>
         </config>
      </attribute>
   </mbean>

</server>
```

Es existieren eingeschränkte Attribute, die nicht verändert werden sollten, wenn Sie sich nicht völlig sicher hinsichtlich der Konsequenzen sind. Die folgenden Attribute können ohne Risiko verändert und entsprechend den Anforderungen Ihres Projekts konfiguriert werden:

clientLeasePeriod

clientLeasePeriod - Clients schicken periodisch *Herzschläge* an den Server, um diesem mitzuteilen, dass sie noch am Leben sind. Erhält der Server nach Ablauf einer bestimmten Zeit keinen solchen Herzschlag ("Heartbeat"), so schließt er die Verbindung und entfernt alle Ressourcen am Server, die der Session dieses Clients entsprechen. Die **`clientLeasePeriod`** bestimmt die Periode von Herzschlägen in Millisekunden. Der Wert ist in Millisekunden. Der Standardwert beträgt **`10000`**.

Standardmäßig schließt der Server einen Client, wenn er innerhalb der doppelten **`clientLeasePeriod`** keinen Herzschlag (Heartbeat) erhalten hat. In Realität wird der Zeitraum automatisch der Systemlast angepasst.

numberOfRetries

Dies korrespondiert wirkungsvoll mit der Anzahl von Sekunden, die JBoss Remoting am Client Connection-Pool blockt und auf Freiwerden einer Verbindung wartet. Falls Sie eine sehr große Anzahl von Sessions haben, die von einem Server gleichzeitig auf den Client zugreifen und Sie Probleme haben, Verbindungen vom Pool zu erhalten, sollten Sie es in Erwägung ziehen, diesen Wert zu erhöhen.

clientMaxPoolSize

JBoss Remoting wartet auf Client-Seite einen Pool von TCP-Verbindungen, an denen Anfragen bedient werden. Falls eine sehr große Anzahl von Sessions gleichzeitig von einem Client auf den Server zugreift und Sie Probleme haben sollten innerhalb einer bestimmten Zeit Verbindungen aus dem Pool zu erhalten, können Sie diesen Wert erhöhen.

secondaryBindPort

Der bisocket Transport verwendet Steuerungsverbindungen ("Control Connections"), um Steuerungsnachrichten zwischen Server und Client weiterzugeben. Dies ist die Adresse, an die das sekundäre **`ServerSocket`** anbindet. Falls Sie hinter einer Firewall arbeiten möchten, so können Sie einen bestimmten Wert hierfür festlegen, je nach Ihrer Firewall-Konfiguration.

secondaryConnectPort

Dies ist der Port den der Client für seine Verbindung verwendet. Es empfiehlt sich dies festzulegen, damit es Clients möglich ist, mit NAT-Routern zu arbeiten.

maxPoolSize

Dies ist die Anzahl der auf Serverseite zum Bedienen von Anfragen verwendeter Threads.

Standardmäßig bindet JBoss Messaging an **`${jboss.bind.address}`**, was durch Ausführung des **`./run.sh -c`** ***`[yourconfig]`*** **`-b`** ***`[yourIP]`***-Befehls definiert werden kann.

Sie können **`remoting-bisocket-service.xml`** ändern, falls Sie einen anderen Kommunikations-Port verwenden möchten.

Warnung

Ändern Sie die Werte in der Connector-Konfiguration nicht, außer den oben angeführten Änderungen können dazu führen, dass JBoss Messaging nicht mehr ordnungsgemäß funktioniert.

5.10. ServiceBindingManager

Der **`SeviceBindingManager`** liefert mehrere Applikationsserverinstanzen, die an unterschiedlichen Portbereichen an derselben IP laufen, was während der Entwicklung hilfreich sein kann. Dies kann auch auf andere Weise erfolgen, aber der **`ServiceBindingManager`** geht vielen Schwierigkeiten aus dem Weg.

Kapitel 6. Clustering Hinweise

Um dabei zu helfen, Clustering-bezogene Informationen zu finden, wird eine Zusammenfassung aller Überlegungen in diesem Teil des Handbuchs samt Links zu den dazugehörigen Komponenten von JBoss Messaging bereitgestellt.

6.1. Eindeutige Server Peer ID

Das Clustering von JBoss Messaging sollte in den meisten Fällen ohne wesentliche Konfigurationsänderungen problemlos funktionieren. Es ist jedoch essentiell, dass jedem Node eine eindeutige Server-ID zugewiesen wird.

Jeder deployte Node muss eine eindeutige ID besitzen, einschließlich derer in einem bestimmten LAN-Cluster sowie derjenigen, die durch Message Bridges (Nachrichtenbrücken) verbunden sind.

Das ***`ServerPeerID`***-Attribut wird zur Einstellung dieser Informationen verwendet. Unter Abschnitt 5.2, „ServerPeer-Attribute“ finden Sie weitere Information.

6.2. Geclusterte Destinationen

JBoss Messaging clustert JMS-Warteschlangen (Java Message Service) und Topics transparent über das Cluster hinweg. An eine distribuierte Warteschlange oder Topic gesendete Nachrichten an einem Node sind an anderen Nodes verwendbar. Um eine bestimmte Destination als geclustert zu designieren, stellen Sie das **`clustered`**-Attribut entsprechend ein. Weitere Informationen finden Sie unter Abschnitt 5.4.1, „PostOffice-Attribute“.

JBoss Messaging balanciert Nachrichten zwischen Nodes, und sorgt dafür, dass schnellere und langsamere Verbraucher effizient die Verabeitungslasten über den Cluster abgleichen.

Falls Sie keine Redistribution von Nachrichten zwischen Nodes wünschen, aber die anderen Eigenschaften geclusterter Destinationen beibehalten möchten, so können Sie dies einrichten, indem Sie das **`ClusterPullConnectionFactoryName`**-Attribut am Server Peer nicht festlegen. Weitere Informationen zu diesem Attribut finden Sie unter Abschnitt 5.2, „ServerPeer-Attribute“.

6.3. Geclusterte dauerhafte Abonnements

Dauerhafte Abonnements von JBoss Messaging können ebenfalls geclustert werden. Dies bedeutet, dass mehrere Abonnementen dasselbe dauerhafte Abonnement von verschiedenen Nodes des Clusters aus verwenden können. Ein dauerhaftes Abonnement wird automatisch geclustert, wenn das Topic geclustert ist.

Weitere Informationen zur Konfiguration geclusterter Topics und Warteschlangen finden Sie im ***`Clustered`***-Attribute in Abschnitt 5.4.1, „PostOffice-Attribute“

6.4. Geclusterte temporäre Destinationen

JBoss Messaging unterstützt außerdem temporäre geclusterte Topics und Warteschlangen. Alle temporären Topics und Warteschlangen werden geclustert, wenn das Post Office geclustert ist.

Weitere Informationen zur Konfiguration geclusterter Topics und Warteschlangen finden Sie im ***`Clustered`***-Attribute in Abschnitt 5.4.1, „PostOffice-Attribute“

6.5. Nicht geclusterte Server

Stellen Sie das geclusterte ***`PostOffice`***-Attribut auf **`false`** ein, falls nicht alle Nodes in einem Cluster teilnehmen sollen oder falls der Server nicht geclustert werden soll.

Weitere Informationen zur Konfiguration nicht-geclusterter Server finden Sie in den verschiedenen Attributen in Abschnitt 5.4.1, „PostOffice-Attribute“.

6.6. Nachrichtenreihenfolge im Cluster

Um sicherzustellen, dass Nachrichten in derselben Reihenfolge verbraucht werden, in der sie erstellt werden, setzen Sie eine strikte JMS-Reihenfolge indem Sie das **`DefaultPreserveOrdering`** Server Peer Attribute auf **`true`** einstellen. Während es auf **`true`** eingestellt ist, können Nachrichten nicht so frei im Cluster verteilt werden. Der Standardwert lautet **`false`**.

6.7. Idempotente Operationen

Eine Nachricht wird garantiert persistiert, wenn eine persistente Destination ohne Ausnahme reagiert.

Eine Ausnahme garantiert nicht, dass die Nachricht *nicht* persistiert wurde, da es zwischen der Persistierung der Nachricht und einer Antwort an den Aufrufer zu einem Fehler gekommen sein kann.

Applikationen müssen daher kodiert sein, damit Operationen *idempotent* sind — das heißt Operationen können wiederholt werden, ohne dass das System inkonsistent wird.

Sie können dieses Verhalten auf Applikationsebene implementieren, indem Sie auf doppelte Nachrichten prüfen und diese verwerfen, wenn die Originalnachricht erfolgreich versendet wurde. Dieses *Prüfen auf Verdoppelung* ist eine leistungsfähige Technick, die die Notwendigkeit von XA-Transaktionen erübrigen kann.

JBoss Messaging ist so konfiguriert, dass es standardmäßig in einer geclusterten Umgebung auf Verdoppelung prüft.

Persistenzüberlegungen finden Sie in Abschnitt 5.2.1, „ServerPeer Methoden“, Abschnitt 5.3, „Die Datenbank wechseln“, Abschnitt 5.5, „Konfiguration des Persistenz-Managers“ und Abschnitt 8.1, „Message Bridge Übersicht“.

6.8. Geclusterte Connection Factories

Wenn **`supportsLoadBalancing`** der Connection Factory auf **`true`** eingestellt ist, so erfolgt bei nachfolgenden Versuchen eine Verbindung zu erstellen ("create connection") ein Round Robin zwischen verfügbaren Servern. Der erste Node, an dem dies versucht wird, wird zufällig ausgewählt.

Ist **`supportsFailover`** auf **`true`** eingestellt, so erfolgt der Failover transparent und automatisch wenn ein Verbindungsfehler aufgespürt wird.

Weitere Informationen zur Konfiguration von Connection Factories finden Sie unter Abschnitt 5.8.1, „ConnectionFactory gemanagte Bean Attribute“.

Kapitel 7. JBoss Messaging XA Recovery-Konfiguration

Dieser Abschnitt beschreibt die Konfiguration von JBoss Transactions zur Handhabung von XA-Recovery für JBoss Messaging Ressourcen in der JBoss Enterprise Application Platform.

Der JBoss Transactions Recovery Manager kann ganz einfach dahingehend konfiguriert werden, dass er kontinuierlich nach JBoss Messaging XA Ressourcen pollt und wieder herstellt, wodurch eine extrem hohe Ebene der Beständigkeit von Transaktionen gewährleistet wird.

Zur Aktivierung des JBoss Transactions Recovery Manager fügen Sie der **`$JBOSS_HOME/server/$PROFILE/conf/jbossts-properties.xml`** eine Zeile hinzu. Der folgende Code-Schnipsel beinhaltet die erforderliche Zeile:

```
<properties depends="arjuna" name="jta">
  <!--
  Support subtransactions in the JTA layer?
  Default is NO.
  -->
  <property name="com.arjuna.ats.jta.supportSubtransactions" value="NO"/>
  <property name="com.arjuna.ats.jta.jtaTMImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.TransactionManagerImple"/>
  <property name="com.arjuna.ats.jta.jtaUTImplementation"

value="com.arjuna.ats.internal.jta.transaction.arjunacore.UserTransactionImple"/>
  <!--
      *** Add this line to enable recovery for JMS resources using
DefaultJMSProvider ***
  -->
  <property name="com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1"

value="org.jboss.jms.server.recovery.MessagingXAResourceRecovery;java:/DefaultJMSPr
ovider"/>

</properties>
```

Im Beispiel oben wird der Recovery Manager versuchen, JMS Ressourcen mittels JMS Provider Loader, **`DefaultJMSProvider`** wiederherzustellen.

`DefaultJMSProvider` wird mit der JBoss Enterprise Application Platform vertrieben. Er ist in **`$JBOSS_HOME/server/$PROFILE/conf/jms-ds.xml`** (oder, in einer geclusterten Umgebung, in **`hajndi-jms-ds.xml`**) definiert. Um die Wiederherstellung mit einem anderen JMS Provider Loader durchzuführen (z.B. einem, der mit einem Remote JMS Provider übereinstimmt), fügen Sie der Properties-Datei eine weitere Zeile hinzu und legen Sie Ihren Remote Provider statt **`DefaultJMSProvider`** fest. Der Name Ihres Providers sollte in der Konfigurationsdatei des gemanagten Beans aufgeführt sein.

Jeder Provider benötigt einen eindeutigen Namen, zum Beispiel **`com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING1`**, **`com.arjuna.ats.jta.recovery.XAResourceRecovery.JBMESSAGING2`**, u.s.w.

Die Recovery sollte auch mit einem beliebigen JMS-Provider funktionieren, der wiederherstellbare XAResources (d.h. ordnungsgemäß **`XAResource.recover()`**) implementiert.

Soll der Recovery Manager zur Wiederherstellung von Transaktionen von einem beliebigen Node des Clusters in der Lage sein, so müssen Sie eine Zeile in **`hajndi-jms-ds.xml`** für jeden Node des Clusters hinzuzufügen.

Kapitel 8. JBoss Messaging Message Bridge Konfiguration

8.1. Message Bridge Übersicht

JBoss Messaging beinhaltet eine vollständig funktionale Message Bridge.

Die Bridge nimmt Nachrichten von einer Quellwarteschlange oder einem Topic und schickt diese an eine Zielwarteschlange oder ein Topic, in der Regel einen anderen Server. Die Quell- und Ziel-Server müssen sich nicht im selben Cluster befinden, was "Bridging" geeignet für das Senden von Nachrichten von einem Cluster an einen anderen macht, etwa über ein WAN hinweg und auch im Falle einer unzuverlässigen Verbindung.

Eine Bridge wird als ein gemanagtes Bean innnerhalb einer JBoss Enterprise Application Platform deployt. Um das Deployment durchzuführen, fügen Sie den Deskriptor des gemanagten Beans dem **`deploy`**-Verzeichnis einer Enterprise Application Platform Konfiguration hinzu, die JBoss Messaging enthält.

Ein Beispiel in **`$JBOSS_HOME/docs/examples/jboss-messaging-examples/bridge/`** demonstriert eine einfache Bridge, die in der JBoss Enterprise Application Platform deployt wird und Nachrichten von der Quell- zur Ziel-Destination verschiebt.

Die Bridge kann auch zur Überbrückung von Nachrichten von anderen nicht JBoss Messaging JMS Servern verwendet werden, so lange diese JMS 1.1 konform sind.

Die Bridge verfügt über eingebaute Ausfallsicherheit, d.h. falls die Verbindung des Quell- oder Ziel-Servers verloren wird, wird die Bridge versuchen eine Verbindung zum Quell- und/oder Ziel wieder herzustellen bis diese wieder online kommen. Ist dies der Fall wird der Betrieb regulär fortgesetzt.

Die Bridge kann so konfiguriert werden, dass sie Nachrichten verwendet, die mit einem bestimmten JMS-Selektor übereinstimmen

Dies kann so konfiguriert werden, das aus einer Warteschlange oder einem Topic gespeist wird. Wird aus einem Topic gespeist, so kann derart konfiguriert werden, das ein dauerhaftes oder nicht dauerhaftes Abonnement verwendet wird.

Die Bridge kann so konfiguriert werden, dass Sie Nachrichten mit einer von drei *quality of service* (QoS) Ebenen (Qualitätsebenen von Service) überbrückt. Diese sind:

Bridge QoS Ebenen

`QOS_AT_MOST_ONCE`

Mit diesem Modus erreichen Nachrichten die Destination von der Quelle höchstens einmal. Die Nachrichten werden an der Quelle entnommen und vor dem Senden an die Destination verwendet. Daher existiert die Möglichkeit, dass es zwischen deren Entnahme von der Quelle und deren Ankunft an der Destination zu einer Fehlfunktion kommt und diese verlorengehen. Daher erfolgt die Lieferung höchstens ein Mal.

Dieser Modus ist sowohl für persistente als auch nicht-persistente Nachrichten verfügbar.

`QOS_DUPLICATES_OK`

Mit diesem Modus werden werden Nachrichten von der Quelle aus verwendet und nach dem erfolgreichen Senden an die Destination bestätigt. Es besteht daher die Möglichkeit, dass bei einer Fehlfunktion nach dem Senden an die Destination diese erneut gesendet werden, wenn das System wieder funktionsfähig ist.

Dieser Modus ist sowohl für persistente als auch nicht-persistente Nachrichten verfügbar.

`QOS_ONCE_AND_ONLY_ONCE`

Dieser Modus legt fest, dass Nachrichten genau einmal ankommen. Sind die Nachrichtenquelle und Destination auf derselben JBoss Messaging Serverinstanz, so kann die Nachricht in

derselben lokalen Transaktion gesendet und empfangen werden.

Befinden sich Quelle und Destination auf unterschiedlichen Servern, so können Sie "Message High Durability" implementieren, indem Sie eine von der JBoss Transactions JTA Implementierung gesteuerte JTA-Transaktion verwenden. Wird JTA benötigt, so müssen beide Connection Factories **`XAConnectionFactory`**-Implementierungen sein.

Dieser Modus ist sowohl für persistente als auch nicht-persistente Nachrichten verfügbar.

Dieser Modus erfordert Protokollierung an sowohl auf Seiten des Transaktionsmanagers als auch der Ressourcen, um die Wiederherstellung zu unterstützen. Falls Sie diese Ebene von QOS benötigen, müssen Sie XA Recovery mit JBoss Transaktionen aktivieren.

Alternative Methoden

Für eine spezifische Anwendung kann es möglich sein, Semantik *einmal (nur einmal)* bereitzustellen, ohne die **`QOS_ONCE_AND_ONLY_ONCE`**-Ebene zu verwenden. Dies kann durch Verwendung des **`QOS_DUPLICATES_OK`**-Modus und darauffolgende Prüfung auf Duplikate an der Destination und deren Entsorgung erfolgen.
Es kann möglich sein, **`QOS_ONCE_AND_ONLY_ONCE`**-Verhalten auf Anwendungsebene zu implementieren indem ein Cache empfangener Nachrichten-IDs aufbewahrt wird, mit dem empfangene Nachrichten verglichen werden. Das Cache wäre nur für einen bestimmten Zeitraum gültig, daher ist diese Vorgehensweise vielleicht nicht ganz so wasserfest, kann aber je nach Anwendung, eine im Einzelfall gute Wahl sein.

8.2. Bridge-Deployment

Eine Message Bridge kann durch Ablegen eines Deskriptors des gemanagten Beans im **`deploy`**-Verzeichnis Ihrer JBoss Enterprise Application Platform Installation, die JBoss Messaging enthält, ganz einfach deployt werden.

8.3. Bridge-Konfiguration

Hier ist ein Code-Beispiel für die Konfiguration einer Message Bridge einschließlich aller Attribute. Einige Attribute wurden für diese Konfiguration auskommentiert, da es unpassend ist, diese alle auf einmal festzulegen.

Beispiel 8.1. Message Bridge Konfiguration

```
<mbean code="org.jboss.jms.server.bridge.BridgeService"
  name="jboss.messaging:service=Bridge,name=TestBridge"
  xmbean-dd="xmdesc/Bridge-xmbean.xml">

  <!-- The JMS provider loader that is used to lookup the source
       destination
  -->
  <depends optional-attribute-name="SourceProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JMS provider loader that is used to lookup the target
       destination
  -->
  <depends optional-attribute-name="TargetProviderLoader">
  jboss.messaging:service=JMSProviderLoader,name=JMSProvider</depends>

  <!-- The JNDI lookup for the source destination -->
  <attribute name="SourceDestinationLookup">/queue/A</attribute>

  <!-- The JNDI lookup for the target destination -->
  <attribute name="TargetDestinationLookup">/queue/B</attribute>

  <!-- The username to use for the source connection
  <attribute name="SourceUsername">bob</attribute>
  -->

  <!-- The password to use for the source connection
  <attribute name="SourcePassword">BobSecur3</attribute>
  -->

  <!-- The username to use for the target connection
  <attribute name="TargetUsername">mary</attribute>
  -->

  <!-- The password to use for the target connection
  <attribute name="TargetPassword">MaryS3cur3</attribute>
  -->

  <!-- Optional: The Quality Of Service mode to use, one of:
    QOS_AT_MOST_ONCE = 0;
    QOS_DUPLICATES_OK = 1;
    QOS_ONCE_AND_ONLY_ONCE = 2;
  -->
  <attribute name="QualityOfServiceMode">0</attribute>

  <!-- JMS selector to use for consuming messages from the source
  <attribute name="Selector">specify jms selector here</attribute>
  -->

  <!-- The maximum number of messages to consume from the source
       before sending to the target
  -->
  <attribute name="MaxBatchSize">5</attribute>

  <!-- The maximum time to wait (in ms) before sending a batch to the
       target even if MaxBatchSize is not exceeded. -1 means wait forever
  -->
  <attribute name="MaxBatchTime">-1</attribute>

  <!-- If consuming from a durable subscription this is the subscription
       name
       <attribute name="SubName">mysub</attribute>
  -->

  <!-- If consuming from a durable subscription this is the client ID to
       use
       <attribute name="ClientID">myClientID</attribute>
  -->
```

```
    <!-- The number of ms to wait between connection retrues in the event
         connections to source or target fail
    -->
    <attribute name="FailureRetryInterval">5000</attribute>

    <!-- The maximum number of connection retries to make in case of failure,
         before giving up -1 means try forever
    -->
    <attribute name="MaxRetries">-1</attribute>

    <!-- If true then the message ID of the message before bridging will be
         added as a header to the message so it is available to the
         receiver. Can then be sent as correlation ID to correlate in a
         distributed request-response
    -->
    <attribute name="AddMessageIDInHeader">false</attribute>

      </mbean>
```

Message Bridge Konfigurationsattribute

SourceProviderLoader, TargetProvider Loader

Das **`JMSProviderLoader`**-gemanagte Bean wird von der Bridge dazu verwendet, die Quell-Connection-Factory und Quell-Destination zu finden. Standardmäßig wird die JBoss Enterprise Application Platform mit einem **`JMSProviderLoader`** geliefert, der in der **`$JBOSS_HOME/server/$PROFILE/deploy/messaging/jms-ds.xml`**-Datei deployt wird und als standardmäßiger lokaler **`JMSProviderLoader`** dient. Bei einer geclusterten Konfiguration spielt **`hajndi-jms-ds.xml`** dieselbe Rolle.

Falls Ihre Ziel-Destination sich auf einem anderen Server befindet oder sogar mit einem anderen, nicht-JBoss JMS-Provider korrespondiert, so können Sie eine weitere **`JMSProviderLoader`** MBean Instanz deployen, die den Remote JMS Provider referenziert und dies von diesem Attribut aus referenzieren. Die Bridge würde dann den Remote JMS Provider verwenden, um die Ziel-Destination zu kontaktieren.

Bitte beachten Sie, dass - falls Sie eine Remote nicht-JBoss Messaging Quelle oder Ziel verwenden und ausschließlich **`QOS_ONCE_AND_ONLY_ONCE`**-Auslieferung (Delivery) wünschen - der Remote JMS-Provider eine vollständig funktionale JMS XA Ressourcen-Implementierung bereitstellen muss, die Remote vom Server aus funktioniert.

SourceDestinationLookup

Dies ist der vollständige JNDI-Lookup für die Quell-Destination unter Verwendung des **`SourceProviderLoader`**, wie **`/queue/mySourceQueue`**.

TargetDestinationLookup

Dies ist der vollständige JNDI-Lookup für die Zieldestination unter Verwendung des **`TargetProviderLocator`**, wie **`/topic/myTargetTopic`**.

SourceUsername

Dieses optionale Attribut legt beim Erstellen der Quell-Verbindung den Benutzernamen fest.

SourcePassword

Dieses optionale Attribut legt beim Erstellen der Quell-Verbindung das Passwort fest.

TargetUsername

Dieses optionale Attribut legt beim Erstellen der Ziel-Verbindung den Benutzernamen fest.

TargetPassword

Dieses optionale Attribut legt beim Erstellen der Ziel-Verbindung das Passwort fest.

QualityOfServiceMode

Dieser ganzzahlige Wert repräsentiert die *gewünschte Qualität des Service*-Modus ("quality of service"). Mögliche Werte sind:

- **0** zur Repräsentation von **QOS_AT_MOST_ONCE**
- **1** zur Repräsentation von **QOS_DUPLICATES_OK**
- **2** zur Repräsentation von **QOS_ONCE_AND_ONLY_ONCE**

Unter Abschnitt 8.1, „Message Bridge Übersicht“ finden Sie eine Erklärung dazu, was diese Modi bedeuten.

Selektor

Dieses optionale Attribut kann einen JMS Selektor-Ausdruck enthalten, der zum Verbrauch von Nachrichten von der Quell-Destination verwendet werden kann. Nur Nachrichten, die mit dem Selektor-Ausdruck übereinstimmen werden von der Quelle zur Ziel-Destination überbrückt. Der Selektor-Ausdruck muss der JMS-Selektor-Syntax folgen, die hier festgelegt ist: http://java.sun.com/j2ee/1.4/docs/api/javax/jms/Message.html.

Für optimale Performance, wenden Sie Selektoren an Quell-Topic-Abonnements zu Quell-Warteschlangen-Verbrauchern an.

MaxBatchSize

Dieses Attribut legt die maximale Anzahl von Nachrichten fest, die von der Quell-Destination verwendet werden soll, ehe diese als Nachrichten-Batch an die Ziel-Destination gesendet werden. Dieser Wert muss größer oder gleich **1** sein.

MaxBatchTime

Dieses Attribut legt die maximale Anzahl von Millisekunden fest, die gewartet werden soll, ehe ein Nachrichten-Batch an ein Ziel gesendet wird, selbst wenn die Anzahl der verwendeten Nachrichten **MaxBatchSize** noch nicht erreicht hat. Der Wert muss entweder **-1** sein, um 'wait forever' ("ewig warten"), oder größer/gleich **1** sein, um eine tatsächliche Zeit festzulegen.

SubName

Repräsentiert den Namen des dauerhaften Abonnements, das vom Quell-Destinations-Topic zehrt.

ClientID

Repräsentiert die JMS Client-ID, die bei der Erstellung oder Suche des dauerhaften Abonnements verwendet wird, das vom Quell-Destinations-Topic zehren wird.

FailureRetryInterval

Dies repräsentiert die Dauer an Zeit (in Millisekunden), die gewartet werden soll, ehe der Versuch unternommen wird, Verbindungen am Quell- oder Ziel-Server wieder herzustellen, wenn festgestellt wurde, dass eine Fehlfunktion vorliegt.

MaxRetries

Dies repräsentiert die Anzahl von Malen, die ein Versuch unternommen werden soll, Verbindungen an die Quell- oder Ziel-Server wieder herzustellen, wenn festgestellt wurde, dass eine Fehlfunktion vorliegt. Die Bridge stellt nach der festgelegten Anzahl von Malen weitere Versuche ein. Der Wert **-1** repräsentiert 'try forever' (d.h. für immer versuchen).

AddMessageIDInHeader

Falls **`true`**, so wird die Message-ID der Originalnachricht der an die Destination gesendeten Nachricht im Header **`JBossMessage.JBOSS_MESSAGING_BRIDGE_MESSAGE_ID_LIST`** angehängt. Wird die Nachricht mehr als einmal "gebridged", so wird die Message-ID angehängt. Dies aktiviert die Verwendung eines distribuierten Anfragen-Antwort Musters.

Kapitel 9. Aktivieren der JBoss Messaging Ordering Group

Dieser Abschnitt beschreibt wie das JBoss Messaging "Ordering Group"-Feature verwendet wird, um eine strenge Nachrichtenreihenfolge zu gewährleisten.

Messaging Ordering Group ist die JBoss Messaging Implementierung von strenger Nachrichtenreihenfolge. Ist das "Ordering Group"-Feature aktiviert, so beeinflussen Nachrichtenprioritäten die Reihenfolge, in der die Nachrichten ausgeliefert werden, nicht mehr. Nachrichten in einer bestimmten Reihenfolgengruppe werden in der exakten Reihenfolge ausgeliefert, in der sie in der Zielwarteschlange (FIFO) ankommen.

Reihenfolgengruppen werden anhand ihres String-Namens identifiziert und folgen folgenden Regeln:

Regel 1

Die Nachrichten, die Teil einer Reihenfolgengruppe bilden, werden eine nach der anderen ausgeliefert. Die nächste Nachricht wird nicht ausgeliefert, ehe die Auslieferung der vorherigen Nachricht nicht abgeschlossen ist. Der Abschluss der Nachrichtenauslieferung wird auf unterschideliche Weise signalisiert, je nach Einstellung des Bestätigungsmodus:

- Der **`CLIENT_ACKNOWLEDGE`**-Modus führt dazu, dass die **`Message.acknowledge()`**-Methode den Status als abgeschlossen signalisiert.
- Die **`AUTO_ACKNOWLEDGE`** und **`DUPS_OK_ACKNOWLEDGE`**-Modi führen dazu, dass der Abschluss der Nachricht als eines der folgenden identifiziert wird:
  - eine erfolgreiche Wiedergabe der **`MessageConsumer.receive()`**-Methoden oder
  - eine erfolgreiche Wiedergabe des **`onMessage()`**-Aufrufs des **`MessageListener()`**.

Hinweis

Wird der Message Consumers geschlossen, so gilt die zum Zeitpunkt der Schließung bearbeitete Nachricht als abgeschlossen. Dies ist unabhängig davon der Fall, ob vor Schließung des Message Consumers **`*_ACKNOWLEGE`** aufgerufen wird oder nicht.

Regel 2

Im Falle des transaktionalen Empfangs von Nachrichten wird die nächste Nachricht nicht ausgeliefert ehe die Transaktion nicht festgeschrieben wurde, was die Empfangsbestätigung der aktuellen Nachricht beinhaltet. Wird die Transaktion zurückgesetzt, so wird die Nachricht abgebrochen, an den JMS-Server zurückgeschickt und für die nächste Auslieferung verfügbar gemacht.

9.1. Wie "Message Ordering Group" aktiviert wird

Die JBoss Messaging Ordering Group kann auf zwei Arten aktiviert werden. Entweder mittels des API oder durch Konfigurationsänderungen.

9.1.1. Aktivierung mit dem API

Um das "Ordering Group" Feature von JBoss Messaging zu nutzen ist es notwendig einen **`JBossMessageProducer`** zu erstellen, wie in folgendem Codebeispiel dargestellt wird:

```
JBossMessageProducer producer=(JBossMessageProducer)session.createProducer(queue);
```

Nachdem er erstellt wurde bietet der JBossMessageProducer zwei Methoden für das Starten und Beenden einer "Ordering Group" (Reihenfolgengruppe).

enableOrderingGroup()

Das folgende Codebeispiel demonstriert, wie eine Reihenfolgengruppe mit dem Namen **`ogrpName`** erstellt wird.

```
public void enableOrderingGroup(String ogrpName) throws JMSException
```

Wird er aufgerufen, so versendet der Producer Nachrichten für die "Ordering Group". Wird ein **`null`**-Parameter bereitgestellt, so wird der Name der "Ordering Group" automatisch generiert. Wird diese Methode mehr als einmal aufgerufen, so werden vorherige Methodenaufrufe außer Kraft gesetzt.

disableOrderingGroup()

Das folgende Code-Beispiel demonstriert, wie die Erstellung von "Ordering Group" Nachrichten gestoppt wird.

```
public void disableOrderingGroup() throws JMSException
```

Nachdem er aufgerufen wurde, stoppt der Producer die Versendung von "Ordering Group" Nachrichten und setzt sein Standardverhalten fort.

9.1.2. Konfigurationsänderungen

Nutzer können eine JBoss Messaging Connection Factory so konfigurieren, dass das "Ordering Group Feature" aktiviert ist. Um dies zu tun, werden zwei neue Attribute der Factory-Service-Konfigurationsdatei hinzugefügt. Diese sind:

- **`EnableOrderingGroup;`**
  - Setzen Sie diese Property auf **`true`**, um das "Ordering Group"-Feature zu aktivieren. Der Standardwert für diese Property lautet **`false`**.
- **`DefaultOrderingGroupName;`**
  - Der Standardname für die "Message Ordering Group". Der Gruppenname wird automatisch generiert, wenn dieses Attribut fehlt.

Hinweis

Nachdem die Konfiguration zur Aktivierung des "Ordering Group"-Features an einer Connection Factory erfolgt ist, werden alle von einem von der Connection Factory erstellten Producer versendeten Nachrichten zu "Ordering Group"-Nachrichten.

Die folgende Beispiel-Konfigurationsdatei für den Factory Service demonstriert, wie das "Ordering Group"-Feature aktiviert wird:

```
<mbean code="org.jboss.jms.server.connectionfactory.ConnectionFactory";
  name="jboss.messaging.connectionfactory:service=ConnectionFactory";
  xmbean-dd="xmdesc/ConnectionFactory-xmbean.xml">
  <depends optional-attribute-name="ServerPeer">
    jboss.messaging:service=ServerPeer
  </depends>
  <depends optional-attribute-name="Connector">
    jboss.messaging:service=Connector,transport=bisocket
  </depends>
  <depends>
    jboss.messaging:service=PostOffice
  </depends>

  <attribute name="JNDIBindings">
    <bindings>
    <binding>/MyConnectionFactory</binding>
    <binding>/XAConnectionFactory</binding>
    <binding>java:/MyConnectionFactory</binding>
    <binding>java:/XAConnectionFactory</binding>
    </bindings>
  </attribute>

  <!-- This are the two properties -->
  <attribute name="EnableOrderingGroup">true</attribute>
  <attribute name="DefaultOrderingGroupName">MyOrderingGroup</attribute>
</mbean>
```

Der Vorteil der Aktivierung des "Ordering Group"-Features durch Konfigurationsänderungen ist die Leichtigkeit, mit der Nachrichtenreihenfolgenfunktionalität erreicht werden kann, ohne dass Code-Änderungen nötig sind.

9.2. Hinweise und Einschränkungen

Beachten Sie folgende Punkte hinsichtlich der "Ordering Group"-Funktionalität:

- Warteschlangen müssen mit dem "Ordering Group"-Feature verwendet werden. Das Feature funktioniert nicht mit Topics.
- Das "Ordering Group"-Feature sollte nicht in Verbindung mit Message Selectors und terminierter Lieferung ("scheduled delivery") verwendet werden.
- Eine Nachricht gilt als vollständig (abgeschlossen), und die nächste Nachricht wird verfügbar, wenn die ursprüngliche Nachricht "dead" oder "expired" ist. Eine "dead" Nachricht wird in die **`DLQ`** verschoben, während eine "expired" Nachricht in die **`ExpiryQueue`** verschoben wird.
- Bei der Verwendung von **`ConnectionConsumer`** wird die Reihenfolge der Nachrichten beachtet. Allerdings steuert **`ConnectionConsumer`** nicht, welche Session die nächste Nachricht erhält.
- Im Falle einer "Distributed Queue" (distribuierten Warteschlange) sollte der Nutzer **`HASingleton`** verwenden, um sicherzustellen, dass die Funktionen des "Ordering Group"-Features ordnungsgemäß funktionieren.

Änderungsverzeichnis

Version 5.1.0-2.400	**2013-10-30**	**Rüdiger Landmann**

Rebuild with publican 4.0.0

Version 5.1.0-2	**2012-07-18**	**Anthony Towns**

Rebuild for Publican 3.0

Version 5-1	**Wed Sep 15 2010**	**Jared Morgan**

Versionsnummer wurde gemäß neuer Versionierungsrichtlinien geändert.
Überarbeitet für JBoss Enterprise Application Platform 5.1.0.GA, einschließlich:
Kapitel 6 und 7 für das Common Criteria 5.1 Projekt wurden umgestellt. Der angewendete Code wurde, wo angemessen, markiert.
JBOSSCC-49 - Integriert alle Kommentare vom JBoss Messaging Guide Rebase.